NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CALCULATION OF BARRIER SEARCH PROBABILITY OF DETECTION FOR ARBITRARY SEARCH TRACKS

By

Wyatt J. Nash

March 2000

Thesis Advisor:	James N. Eagle
Second Reader:	Lyn R. Whitaker

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2000	Master's Thesis

4. TITLE AND SUBTITLE
Calculation Of Barrier Search Probability Of Detection For Arbitrary Search Tracks

5. FUNDING NUMBERS

6. AUTHOR(S)
Nash, Wyatt J.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*
The Surface Warfare Development Group is responsible for conducting the Ship Anti-submarine Warfare Readiness/Effectiveness Measuring program. They currently employ a standard set of measures for evaluating the performance of shipboard anti-submarine warfare sensors. This research investigates several new performance-based measures to determine if they are more suitable than the standard measures for evaluating the conduct of anti-submarine warfare barrier searches. The investigation simulates barrier searches to determine probability of detection, calculates the proposed measures, and compares the two. The results indicate that the proposed measures can be improved. A barrier search algorithm exploiting target-relative space ideas is developed which generalizes the classical search theory results for predicting probability of detection during barrier search.

14. SUBJECT TERMS
Surface Warfare Development Group (SWDG), Shipboard Anti-submarine Warfare Readiness/Effectiveness Measuring Program (SHAREM), Barrier Search, Modeling and Simulation, Java

15. NUMBER OF PAGES
103

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

CALCULATION OF BARRIER SEARCH PROBABILITY OF DETECTION FOR ARBITRARY SEARCH TRACKS

Wyatt J. Nash
Lieutenant, United States Navy
B.S., University of New Mexico, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

Author: _____
Wyatt J. Nash

Approved by: _____
James N. Eagle, Thesis Advisor

_____
Lyn R. Whitaker, Second Reader

_____
Richard E. Rosenthal, Chair
Department of Operations Research

ABSTRACT

The Surface Warfare Development Group is responsible for conducting the Ship Anti-submarine Warfare Readiness/Effectiveness Measuring program. They currently employ a standard set of measures for evaluating the performance of shipboard anti-submarine warfare sensors. This research investigates several new performance-based measures to determine if they are more suitable than the standard measures for evaluating the conduct of anti-submarine warfare barrier searches. The investigation simulates barrier searches to determine probability of detection, calculates the proposed measures, and compares the two. The results indicate that the proposed measures can be improved. A barrier search algorithm exploiting target-relative space ideas is developed which generalizes the classical search theory results for predicting probability of detection during barrier search.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they are not considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS, ACRONYMS AND/OR ABBREVIATIONS

A	Search Area
L	Barrier Width
N	Times a Search Cell is Covered
Pd	Probability of Detection
SW	Sweep Width
u	Target Speed
v	Searcher Speed
ASW	Anti-submarine Warfare
FLTASWIP	Fleet ASW Improvement Program
MOE	Measure of Effectiveness
MOP	Measure of Performance
OPAREA	Operation Area
SHAREM	Ship Anti-submarine Warfare Readiness/Effectiveness Measuring Program
SURTASS	Surveillance Towed Array Sensor System
SWDG	Surface Warfare Development Group
TDA	Tactical Decision Aid
USW	Undersea Warfare
hr	Hour
kt	Nautical Mile per Hour
nm	Nautical Mile
yd	Yard

ACKNOWLEDGEMENT

The author would like to thank the many people who helped make this thesis possible. First to Dr. James Eagle, who fostered my interest in search theory and motivated me to move beyond simulation and modeling to discover new ideas. To Dr. Arnold Buss for his direction in the use of the SIMKIT simulation package and because he taught simulation theory with infectious enthusiasm. To John Seeley, whose commitment to the Surface Warfare Development Group and to surface fleet anti-submarine warfare continues to this day. Finally to my wife Angelica, who loves me and continues to support me in all of my endeavors.

EXECUTIVE SUMMARY

The Surface Warfare Development Group (SWDG) is responsible for conducting the Ship Anti-submarine Warfare Readiness/Effectiveness Measuring (SHAREM) program. This program is an ongoing series of anti-submarine warfare (ASW) search exercises conducted to aid in the testing of new equipment and search tactics.

The overall objective of the SHAREM program is to collect and analyze high-quality data to quantitatively assess surface ASW readiness and effectiveness. To aid in the analysis of the data, a standard set of measures is used. Although the SHAREM program successfully employs the standard measures to evaluate sensor performance, it is not as successful at using them to evaluate how well a search unit executes a search tactic. In November 1998 the Fleet ASW Improvement Program working group addressed the shortcomings of the standard measures.

Several new performance-based measures are proposed to SWDG by Analysis and Technology of Chesapeake, VA to solve the problem of using the standard measures to evaluate tactics. The proposed measures are part of an evaluative tactical decision aid (TDA) that is used in planning ASW barrier search events and in post-event analysis.

This thesis is a review of two of the proposed measures. After determining that the proposed measures could be improved, a new algorithm is developed for calculating instantaneous probability of detection (Pd) of a transiting target attempting to cross a barrier being patrolled by a searcher following an arbitrary search track.

The purpose of the proposed measures is to quantify how well a searcher executes an ASW barrier search tactic. The underlying idea behind the method used to evaluate the proposed measures is that, if the measure in question is a good one, then there should be a strong relationship between the measure and theoretical Pd.

Bernard Koopman's OEG 56 is a compilation of the work that was done during World War II in support of the ASW effort. It includes formulas for calculating theoretical Pd of a transiting submarine attempting to cross a barrier being patrolled by a searcher following "back-and-forth" or "crossover" search tracks. Because actual search tracks rarely resemble these idealized tracks, Koopman's formulas for calculating

theoretical Pd are not used directly for evaluating the proposed measures. Instead, simulated Pd is used.

By using a Monte Carlo simulation, implemented in Java™, a search track is run repeatedly against random targets and simulated Pd is calculated. Prior to running the simulation, two of the measures are calculated for the search track. Both measures are calculated based on the fraction of the search area that is covered by the searcher. The result is one data point on a plot of measure versus simulated Pd. By repeating the simulation with different search tracks, enough data points are obtained to determine the relationship between each of the measures and simulated Pd using regression analysis. The results of this analysis suggest that the proposed measures can be improved.

There are two reasons why the proposed measures do not appear to have a strong relationship with simulated Pd. First, they do not account for search track orientation or target motion. Second, they are calculated using irrelevant portions of the search area. Both of these problems are eliminated by determining the area covered by the searcher in target-relative space and using a different method for calculating one of the measures. This new method yields theoretical Pd values for a single target that crosses the barrier during the execution of the search track. Insight gained from modeling in target-relative space forms the basis for the general barrier search algorithm that is developed to calculate instantaneous Pd for a continuous stream of targets crossing the barrier.

Koopman introduced the idea that theoretical Pd is the ratio of the area actually searched to the total area to be searched for one half cycle of a search track in target-relative space. His formulas require a repeating track composed of straight legs. The track must repeat so that a half cycle may be defined and Pd calculated. The legs must be straight because the formulas are the analytic solutions to geometry problems that are not easily solved, if at all, for a curved path.

The general barrier search algorithm is based on Koopman's idea. To allow arbitrary search tracks to be evaluated, the algorithm discretizes the target-relative search track and calculates the fraction of the discrete sections of the search track that are covered by the searcher. Each fraction results in a theoretical Pd for a single target that started to cross the barrier at a given time in the past. The continuous calculation of these

theoretical values of Pd as the search progresses results in instantaneous values of barrier search Pd for an arbitrary search track of any length.

The general barrier search algorithm is quite simple and makes a number of assumptions that are not realistic, but produce results that can be verified and validated using analytic solutions from search theory. It does, however, provide a base from which to expand. Additional measures can be added to determine how much search effort is wasted by searching outside of the assigned area, the sensor coverage can be modified and detection rate models can be used to model non-ideal sensors, and search areas can be combined to calculate measures for multiple searchers. Any or all of these improvements can be added based on the desired use of the algorithm.

I. INTRODUCTION

World War II search theory focuses primarily on detecting surfaced U-boats either visually from aircraft or with radar. Koopman's OEG 56 [Ref. 1] is a compilation of the work that was done during the war in support of the anti-submarine warfare (ASW) effort. This includes formulas for calculating the probability of detection (Pd) of a transiting submarine attempting to cross a barrier being patrolled by a searcher following "back-and-forth" or "crossover" search tracks. These formulas are still in use today [Ref. 2].

This thesis is as a review of several new proposed measures that are part of an evaluative tactical decision aid (TDA) for planning ASW exercises and in post-exercise analysis. The TDA assists planners and operators in the design and execution of effective searches. This thesis reviews two of the proposed measures and introduces a new algorithm for calculating Pd of a transiting submarine attempting to cross a barrier being patrolled by a searcher following an arbitrary search track.

A. THE SHIP ANTI-SUBMARINE WARFARE READINESS / EFFECTIVENESS MEASURING PROGRAM

1. Objective and Goals

Since World War II, the development of quieter diesel submarines with greater submerged endurance makes detection more difficult and poses a continuous challenge to the designers and operators of active and passive sonar systems. Present day ASW exercises conducted by the Surface Warfare Development Group (SWDG) aid in the testing of new equipment and search tactics.

The Ship Anti-submarine Warfare Readiness/Effectiveness Measuring (SHAREM) program is an ongoing series of exercises conducted by SWDG. These exercises are conducted in regions of the world's oceans that have tactical significance to the security of the United States. The overall objective of the SHAREM program is to

collect and analyze high-quality data to quantitatively assess surface ASW readiness and effectiveness. One of the goals for achieving this objective is to identify and develop solutions for tactical problems connected with employment of surface ASW systems. [Ref. 3]

2. SHAREM Exercises

Traditionally, the SHAREM exercises support this goal by determining the detection capabilities of new systems. Shortly after the Cold War, the focus of the exercises shifted from equipment evaluation to the evaluation of tactics. The SHAREM program evaluates equipment and tactics using two main types of ASW events – area search and barrier search. The area search event is designed to search for a threat submarine loitering in a 5000 to 8000 square nautical mile area and lasts 12 to 18 hours. The barrier search event is designed to counter a threat submarine that is approaching and attempting to transit a chokepoint and lasts 24 hours.

In order to minimize the number of variables affecting the search, all exercises are conducted in a similar manner. The search assets consist of an SQS-53 sonar equipped ship, an SQS-56 sonar equipped ship, an allied ship, a Surveillance Towed Array Sensor System (SURTASS) ship, a nuclear attack submarine, and four helicopters (one of which has a dipping sonar). These assets conduct structured event runs where they have knowledge of the target's track. This is done so that they may approach the target and gain detection. The average detection range for each searcher is determined during the structured runs and is used in the analysis of the actual search events.

Execution of the actual search events takes place over several days and data such as ship's position, time of target detection, and target solution are gathered. These data are used to reconstruct the events, compare the search data to actual target position data, and evaluate how well the equipment and the ship performed.

B. ASW MEASURES OF EFFECTIVENESS AND PERFORMANCE

1. Standard Measures

Standard Measures of Effectiveness (MOEs) and Measures of Performance (MOPs) are used for the design, conduct, and analysis of ASW system evaluations and exercises [Ref. 4]. These measures include two force-level MOEs and 17 system/platform level MOPs of which only three apply to the detection problem and the remaining apply to classification, localization, attack, and vulnerability. Only the detection MOPs are relevant here because the SHAREM exercises are mainly concerned with the detection problem. The MOEs are the probability that ASW forces accomplish their mission and the probability that threat submarines fail to accomplish their mission. The detection MOPs are the probability of detection as a function of lateral range, the cumulative probability of detection as a function of range, and the figure of merit. The instruction governing the standard measures provides guidance on calculating all of the MOPs and using them to calculate the MOEs.

Although the SHAREM program successfully uses the standard measures to evaluate sensor performance, it is not as successful at using these measures to evaluate how well a unit or force executes an ASW search tactic. For example, a ship may have a high probability of detection as a function of lateral range because of favorable acoustic conditions in the search area, but execute a poor search by remaining stationary within the search area. Although the measure provides information about how well the equipment performs in that situation, it does not describe how well the tactic is executed using the equipment.

In November 1998 the Fleet ASW Improvement Program (FLTASWIP) working group addressed the shortcomings of the standard measures [Ref. 4]. The undersea warfare (USW) tactical development agencies present at the working group concluded that the standard measures must be improved to capture unit and force tactical effectiveness.

2. Performance-based Measures

Several new performance-based measures are proposed to SWDG by Analysis and Technology of Chesapeake, VA to solve the problem of using the standard measures to evaluate tactics. These measures, described in Chapter II, are part of a TDA which attempts to quantify how well unit or force executes an ASW search tactic. The measures are:

- Unit Barrier Probability of Success
- Percent Effective Barrier Versus Assigned Station Area
- Percent Effective Barrier Versus Geographic Area Searched
- Percent Effective Barrier Versus Total Area Searched
- Percent Of Wasted Search Effort
- Force Barrier Probability Of Success

These new measures have been used to evaluate recent SHAREM exercises, but there has been no work done to verify the TDA used to produce them or to determine whether the measures are valid.

C. METHODOLOGY

This thesis consists of two main parts. The first part, Chapter II, describes the proposed measures and compares two of the measures to Pd computed from a barrier search simulation. The observed relationship between the measures and the simulation Pd is weak. Chapter III introduces the classical search theory idea of using area covered in target-relative space to compute barrier search Pd. Using this idea, a considerably better relationship between one of the measures and the simulation Pd is obtained. The second part, Chapter IV, uses insight gained from the results of Chapter III to describe a new algorithm for calculating the barrier search Pd for arbitrary search tracks. Chapter V suggests further development of the algorithm and asserts that Pd calculated using the

algorithm is a more appropriate measure than the Unit Barrier Probability of Success for use in a TDA.

THIS PAGE LEFT INTENTIONALLY BLANK

II. ANALYSIS OF THE SWDG EVALUATIVE TDA

The SWDG evaluative TDA, referred to as the TDA, calculates the proposed measures from data gathered during the structured and actual runs of the SHAREM barrier search events. The first part of this chapter describes the current implementation of the TDA. Parameters that characterize the search event are calculated first followed by the measures that result from the search. Additionally, the key assumptions and terms related to the TDA are described. This description is essential to understanding how the TDA is implemented for analysis.

Although the TDA is available, it is not used to calculate the proposed measures for the analysis. Because the model used for the analysis relies heavily on discrete event simulation of barrier searches, the functions to calculate the measures are included in the simulation model. To differentiate it from the TDA, the model used to analyze the measures is referred to as the simulation. The second part of this chapter describes, in detail, the reasons for using and the development of the simulation.

After defining the simulation scenarios, the third part of this chapter presents the results of the simulation and offers a possible explanation for the results in the context of the proposed measures.

A. CURRENT IMPLEMENTATION

Each SHAREM exercise is broken up into three distinct phases – planning, execution, and analysis. The following is a description of how the TDA is used during the analysis phase and assumes that the planning and execution phases are complete and the data from the exercise are available.

1. Definitions and Initial Calculated Parameters

Before the measures are calculated for a barrier search event, initial parameters that characterize the event are calculated. The parameters are calculated using the

assumed or actual speed of the target submarine and the search tracks. Omni-directional sweep width (SW) is defined for each searcher as twice his average detection range. The average detection range is calculated using the observed detection ranges from the structured runs. Sweep width is divided by target speed (u) to define the analysis period for each searcher. The entire barrier search event is divided up into time segments each of which is equal to the analysis period.

The operational area (OPAREA) that forms the ASW barrier is divided up into geographic search areas that are the assigned stations for the units that comprise the search force. Within his area, each searcher moves about conducting an active sonar search in an attempt to detect the target that is transiting through the chokepoint covered by the OPAREA.

As shown in Figure 1, the TDA partitions the search area into square, typically 500 yd by 500 yd, areas called cells. Because the searcher is allowed to move to the edges of his assigned station, he can "see" targets that are within one detection range of these edges. The cells that exist outside of the search area contain information about the search effort that occurs in the one-detection-range border surrounding the area.

When an active sonar ping occurs, the time of the ping is recorded for all cells that are within the detection range of the searcher relative to his geographic position. After all of the pings that have occurred during the current analysis period have been processed in this way, the ping times for each cell are reviewed to determine if consecutive pings have occurred.

A 6-ping history is used to determine if a cell has been effectively searched. SWDG provides limited background on the 6-ping model. Regardless of the active sonar used, this model assumes that six consecutive pings in a cell results in a 50% probability of detection if the target is present in that cell. [Ref. 5]

A cell that receives six consecutive pings is considered "covered". If the same cell receives another six consecutive pings, it is considered covered twice. A cell that is covered twice during the analysis period is considered "effective". For the purpose of calculating the measures, no distinction is given to a cell that is covered three or more times. It is just another effective cell.

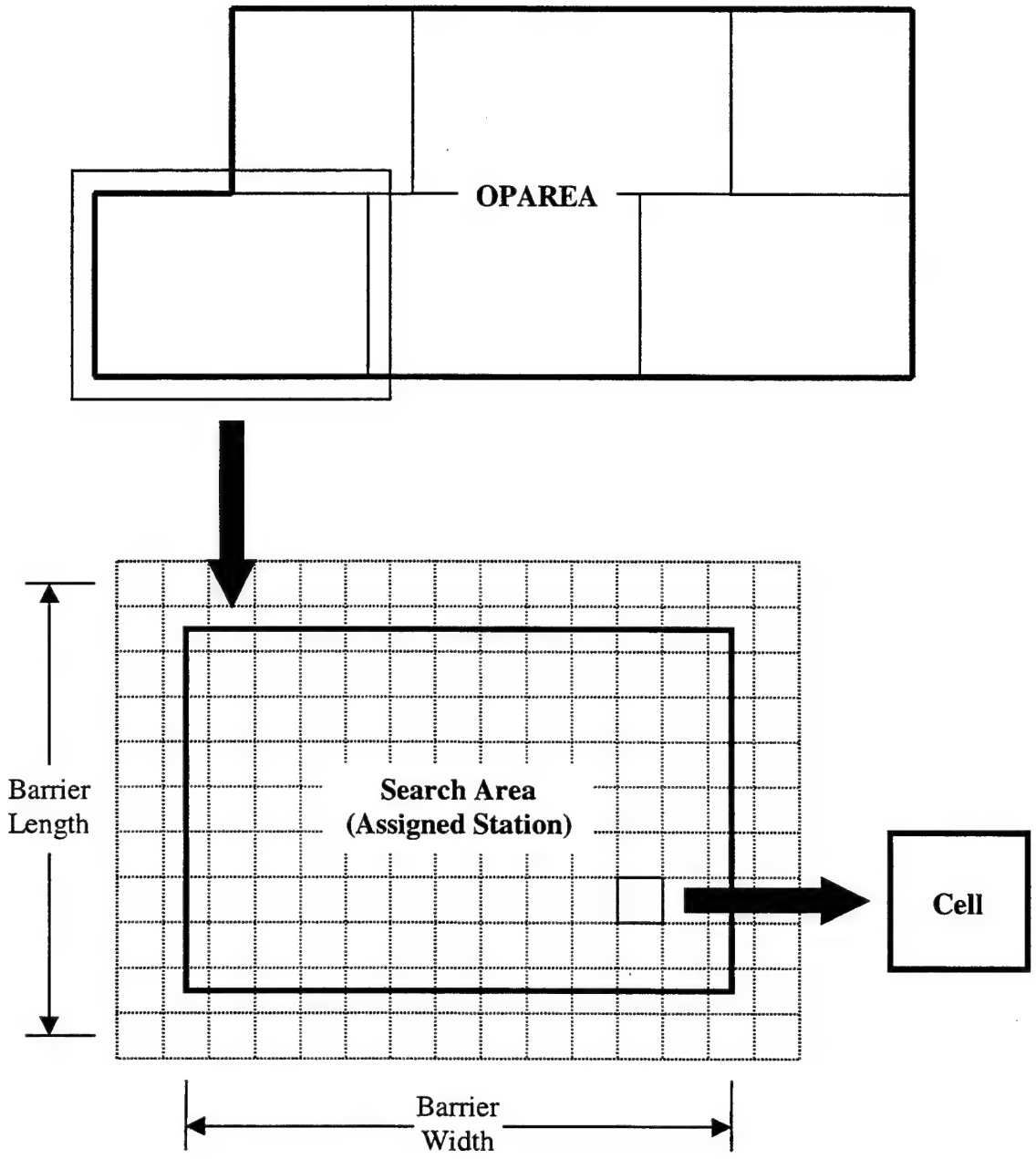

Figure 1. Partitioning of the Search Area

Figure 2 shows the cell coverage for a searcher using a 6-ping history, omni-directional sensor. The search track shown allows the searcher to cover about half of the cells in his assigned station at least once.

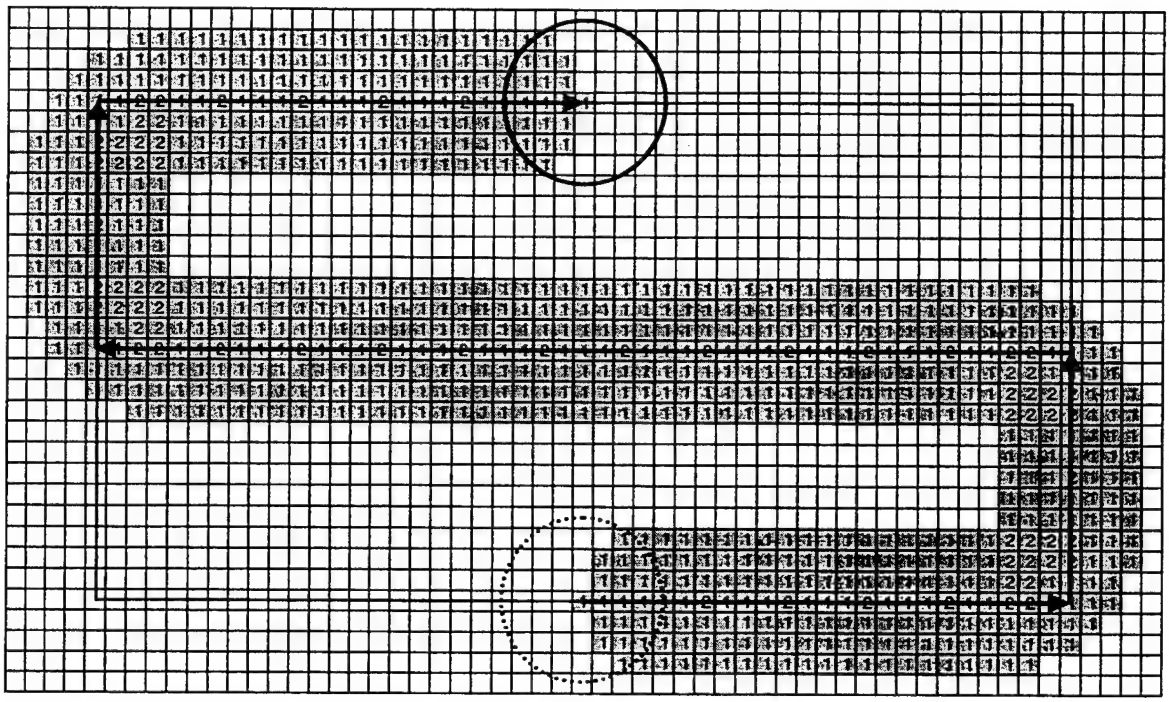

Figure 2. Cell Coverage Using a 6-ping History Omni-directional Sensor

2. Proposed Measures

Once the TDA processes a search track and identifies the number of times each cell in the search area has been covered, the measures are calculated. All of the proposed measures are calculated, directly or indirectly, based on the number of times each cell is covered (N). The two proposed measures of interest are calculated using the search track of Figure 2 and its resulting cell coverage data from Table 1.

Times Cell Covered	Cells Inside Station	Cells Outside Station
0	654	Not required
1	503	156
2	68	0
Total	1225	156

Table 1. Cell Coverage Information for Calculating the Proposed Measures

- Unit Barrier Probability of Success

Measure 1 is the ratio of a weighted sum of the cells inside the assigned station that are covered once, twice, etc., to the total number of cells inside the station. The weights assigned to the cells are calculated as 1 - exp(-N). This results in weights of 0.63, 0.86, and 0.95 for cells covered one, two, and three times, respectively. Using Table 1 data, Measure 1 is $[(0.63)503 + (0.86)68] \div 1225 = 0.31$.

- Percent Effective Barrier Versus Assigned Station

Measure 2 is the ratio of the effective cells inside the assigned station to the total number of cells inside the station. Because an effective cell is one that is covered two or more times, Measure 2 is $(68 + 0) \div 1225 = 0.06$.

- Percent Effective Barrier Versus Geographic Area Searched

The geographic area searched is defined as the area containing all cells that have received at least one active sonar ping. This area may contain cells that are not even covered once. Measure 3 is the ratio of the effective cells inside the assigned station to the number of cells inside the geographic area.

- Percent Effective Barrier Versus Total Area Searched

The total area searched is defined to be the weighted sum of all cells. The weight that is applied to each cell is the number of times the cell is covered. Measure 4 is the ratio of the effective cells inside the assigned station to the total area searched.

- Percent Of Wasted Search Effort

Measure 5 is the ratio of the area outside the OPAREA to the area searched by the unit of interest. This measure can be calculated using either geographic or total area.

- Force Barrier Probability Of Success

Measure 6 is the same as Measure 1 except it is calculated for the entire force instead of a single unit.

B. AN IMPLEMENTATION FOR CONDUCTING ANALYSIS

1. Linking Pd and Measures

The purpose of the proposed measures is to quantitatively assess how well a searcher performs a particular search. The underlying idea behind the method used to evaluate the proposed measures is that, if the measure in question is a good one, then there should be a strong relationship between the measure and Pd of a target that is attempting to cross the barrier. The TDA is designed to calculate the measures, but cannot calculate Pd because the actual search tracks from the exercises do not resemble tracks that have analytic solutions [Refs. 1 and 2].

Using Monte Carlo simulation, a search track is run repeatedly against random targets and Pd is calculated. Prior to running the simulation, the measures are calculated for the search track. The result is one data point on a plot of measure versus Pd. By repeating the simulation with different search tracks, enough data points are obtained to determine the relationship between each of the measures and Pd using regression analysis.

The simulation is implemented in Java™ using the SIMKIT discrete event simulation package [Ref. 7]. The core of the simulation generates search tracks within the search area and target tracks that cross the barrier. Two separate simulation entities are used to gather information about the simulation. They use the listener pattern that is incorporated into SIMKIT to calculate cell coverage and Pd. A SIMKIT listener is a simulation entity that only takes action when a specified event occurs in another simulation entity. Specifically, when an active sonar ping occurs during the simulation, the simulation entity responsible for determining cell coverage updates the number of pings in the appropriate cells of the partitioned search area. A separate Java™ class calculates the measures using the cell coverage information.

2. Simulating a Barrier Search

Appendix A contains a listing of the source code for the class files used to analyze the TDA. The functionality of each class is described in the following explanation of the simulation development. The class Simulation is the main program that controls the execution of the simulation. Simulation sets all of the simulation parameters, instantiates the required simulation entities, executes the simulation, and writes the calculated measures and Pd values to a file.

The simulation is developed in stages and the results of each stage are verified to the greatest extent possible before proceeding. The TDA assumes that the direction from which the target approaches the barrier, known as the threat axis, and the target speed are known, but the point at which the target penetrates the barrier is not. Consequently, all simulated searches used to evaluate the measures are run against targets whose penetration points are uniformly distributed across the width of the barrier and cross at a constant speed perpendicular to the width of the barrier.

Initially, a simple back-and-forth barrier search is run where the searcher patrols back-and-forth across the width of the barrier. The results are compared with those obtained using equation 1.3-3 of [Ref. 2] to verify that the detection algorithms in SIMKIT are correct. Table 2 shows the results for an L=20 nm barrier and a v=12 kt searcher with a SW=4000 yd sensor. All theoretical Pd values are within the 95% large-sample confidence interval.

Target Speed (kts)	Pd_{theory}	$Pd_{1000\ runs}$
4	0.3162	0.313 ± 0.029
6	0.2236	0.222 ± 0.026
12	0.1414	0.145 ± 0.022

Table 2. Pd Verification for a Back-and-forth Barrier Search

In conducting an actual barrier search, it is not possible to achieve perfect back-and-forth motion across the width of the barrier. When planning a barrier search, each

search asset is given an area of the barrier to cover vice a linear segment along the barrier. Within that area, the searcher is free to conduct his search as he deems appropriate. Based on actual ship track data from SHAREM exercises, the evaluation of the measures is conducted assuming that the searcher executes a random search of the area.

Prior to running the simulation against a transiting target, it is run against a stationary target uniformly distributed within the search area. This is the classic area search problem described by Washburn [Ref. 2] and is used to verify that the searcher's movement algorithm is truly random. The random search portion of the simulation is developed in three stages: node search, perfect reflection, and diffuse reflection.

Random node search starts the searcher uniformly within the search area. Another uniformly distributed point is picked within the area and the searcher moves toward the destination at a constant speed. McNish [Ref. 8] proves that this type of search will over-search the center of the area.

To improve the randomness of the search algorithm, the searcher moves in a uniformly distributed direction from his random starting location until he encounters an area boundary. When the boundary is reached, the searcher undergoes specular reflection, meaning the angle of incidence is equal to the angle of reflection. This method, in an attempt to not over-search the center of the area, actually over-searches the edges [Ref. 8].

The final stage in the development of the random area search algorithm is to model reflection off of the area boundary not as specular, but as diffuse. McNish [Ref. 8] provides a detailed description of diffuse reflection and gives the formula for the cosine distribution. The direction in which the searcher will travel when he reflects off of a boundary is determined by taking the inverse of the cosine distribution and using a uniform random number as it's input. The class RandomMoverManager3 implements the diffuse reflection random area search. This class listens to the search entity, and when the searcher stops moving, the inverse cosine function is used to determine the new direction of motion. The searcher moves in that direction until he again encounters an area boundary and quits moving. Table 3 compares the results of the random area search

using the diffuse reflection algorithm for an A=400 nm^2 area and a SW=2000 yd sensor to those obtained using Washburn [Ref. 2]. All theoretical Pd values are within the 95% large-sample confidence interval.

Search Speed (kts)	Search Duration (hrs)	Pd_{theory}	$Pd_{1000\ runs}$
5	10	0.1175	0.110 ± 0.019
10	10	0.2212	0.203 ± 0.025
10	20	0.3935	0.394 ± 0.030
20	20	0.6321	0.612 ± 0.030

Table 3. Pd Verification for a Random Area Search

Modeling a transiting target attempting to cross an area requires a change to the analysis period. The TDA defines the analysis period as the amount of time it takes the target to transit a distance equal to the sweep width of the sensor. This is done because if an omni-directional sensor can make at least one back-and-forth sweep across the width of the barrier in that amount of time, then the target is always detected. For a searcher executing this motion, the current definition of the analysis period is adequate.

The purpose of the TDA, however, is to provide insight into searches conducted with arbitrary motion within an area. Therefore, an appropriate definition for analysis period is the amount of time it takes the target to transit from one detection range above the top of the barrier to one detection range below the bottom of the barrier. This distance is the definition of barrier length as shown in Figure 1. The analysis period is equivalent to the total amount of time that a target has the possibility of being within range of the sensor. The class TargetMoverManager generates constant speed targets that uniformly penetrate the barrier and controls the movement of the targets as they cross the barrier from top to bottom.

3. Calculating the Proposed Measures

The model discussed so far only determines Pd. In order to calculate the proposed measures, a means of updating the cells is required. The TDA determines if consecutive pings have occurred in each cell by reviewing the ping times for the cells after the analysis period ends. Unlike the TDA, the simulation determines the number of consecutive pings and the number of times each cell is covered continuously during execution.

The class PingCellMatrix1 is instantiated by Simulation to model the partitioning of the search area into square cells. The cells are referred to as ping cells because they keep track of the number of pings that have occurred. PingCellMatrix1 is responsible for creating a matrix of SixPingCell instances, creating a masking matrix of type boolean, and updating the ping cell matrix with the masking matrix each time a ping occurs.

Each instance of the class SixPingCell keeps track of how many pings have occurred in that cell. When six consecutive pings have occurred, the number of times the cell has been covered is incremented.

The masking matrix is based on the type of sensor to be modeled. For an omni-directional sensor, all of the masking matrix elements that are within one detection range of the center element in the matrix are assigned a value of true, while all others are false. Baffle regions[1] for the active sonar are considered in the TDA, but the added complexity of modeling these regions precludes their use here.

When a ping occurs during the simulation, the instance method doPing() of the class PingCellMatrix1 determines the current position of the searcher and his corresponding position within the ping cell matrix. This amounts to mapping the searcher's coordinates to the nearest row and column of the matrix. The center element of the masking matrix is then overlaid on the searcher's position in the ping cell matrix. The instances of SixPingCell around the searcher are then updated by scanning the masking matrix and looking for a value of true. When a true value is found, the instance method

[1] The baffle region for sonar is an area where sound cannot be detected due to the physical layout of the sonar system within the ship.

incrementPings() of the appropriate SixPingCell updates the number of pings that have occurred in that cell.

Once the search track is completed and the ping cell matrix reflects all of the pings that have occurred during the search, the measures are calculated. The class Measures has the method getMeasures() that Simulation calls to obtain the information about the search. Only Measures 1 and 2 are calculated and returned.

C. RESULTS

The simulation performs runs of 100 search tracks, each of which performs a different random search of the area, against 30 target tracks that uniformly penetrate the barrier. Initially, four runs are performed using combinations of v=8 and 16 kt searchers with SW=4000 and 8000 yd sensors against a u=4 kt target. The results of these runs suggest that there is a large amount of variability in Pd for a given value of the measure.

A special test scenario is used to compare the variability of the proposed measures to methods and measures described in Chapter III. The scenario uses a v=12 kt searcher with a SW=16000 yd sensor against a u=4 kt target in a 12 nm by L=24 nm area. This choice of search parameters produces a wide range of values for the measures. The results of the test run are shown in Figures 3 and 4.

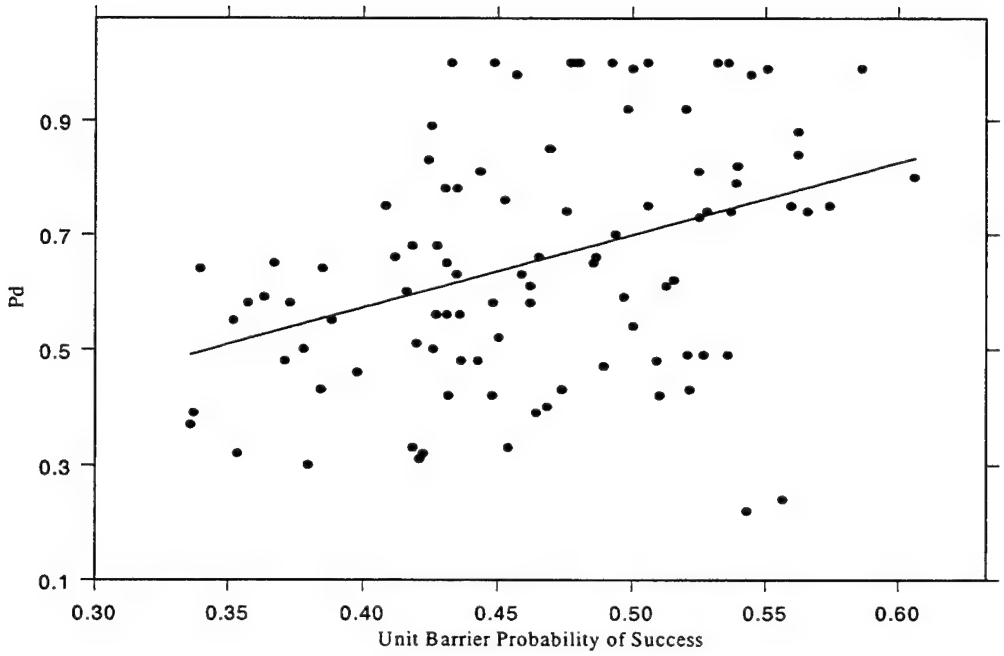

Figure 3. Least-Squares Regression for Proposed Measure 1

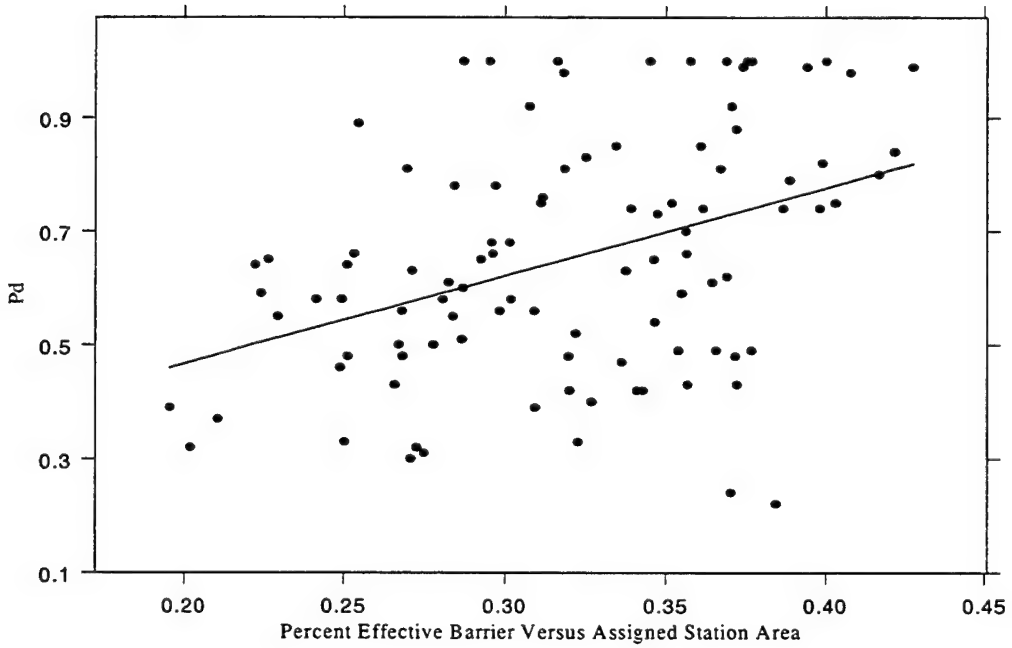

Figure 4. Least-Squares Regression for Proposed Measure 2

A least-squares linear regression of the measure versus Pd data does show a positive relationship, however, there is a large amount of variability in Pd for any given value of the measures. A reasonable explanation for this variability is in the way that the measures are calculated. Both of the measures use some form of summation of the ping cells that are covered during the search without regard to the orientation of the search track. It is possible for a searcher whose motion is in the same direction as the target's to have the same value of proposed measure as a searcher whose motion is perpendicular to the target's. The first searcher, however, will have a lower Pd against a uniformly penetrating target than the second.

Because of the variability in Pd, proposed Measures 1 and 2 do not appear to provide a good quantitative assessment of how well the searcher performed the search within his assigned station. If the variability is actually due to the orientation of the search track, then the proposed measures may provide a better assessment of search effectiveness if the assigned station constrains the searcher to mainly back-and-forth motion across the width of the barrier, perpendicular to the direction of target motion. This aspect is not investigated because a more general solution is desired.

This chapter began by discussing how the SWDG evaluative TDA calculates the proposed measures for ASW barrier searches. The TDA is modeled using a discrete event simulation which calculates Pd for random search tracks against targets that uniformly penetrate the barrier. The simulation also calculates the measures in order to compare them to Pd. Simulation results suggest that the proposed measures can be improved.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROPOSED CHANGES TO THE SWDG EVALUATIVE TDA

The results of Chapter II suggest that the proposed measures can be improved. This chapter investigates possible changes to Measures 1 and 2 with more attention given to Measure 1. The first part of this chapter describes changes that are made to the simulation model to account for search track orientation and target motion. This is done by updating the area covered by the active sonar search in target-relative space. Additional simplifying assumptions are also made. The second part of this chapter presents the results of the simulation and a new method for calculating Measure 1. This new method yields Pd values for a single target crossing a barrier searched with an arbitrary search track. Insight gained in this chapter forms the basis for the general barrier search algorithm that is developed in Chapter IV.

A. MODELING IN TARGET-RELATIVE SPACE

1. Limitations of Geographic Space

When searching for a stationary target or a target whose speed is much smaller than the speed of the searcher, it is probably acceptable to use the method of Chapter II to determine how much of the search area has been covered. For a moving target whose speed is close to that of the searcher, a different approach is recommended if we wish to estimate Pd as the fraction of the search area covered by the searcher.

It is beneficial to visualize the amount of search effort from the target's perspective. Imagine that the target has a display where he is at the center and the amount of area covered by the active sonar search is displayed relative to his position. Assume the searcher is moving down the search channel at the same speed and on a parallel course with the target. On the target's display, the searcher repeatedly covers the same area. Using the methods of Chapter II, the searcher would generate a larger and larger value of the proposed measures, although he is actually not searching any new area

relative to the target. By not adjusting for the speed of the target, the proposed measures overestimate Pd when there is little relative motion between the target and the searcher.

Now the searcher reverses his course. On the target's display, he will see the searcher move up the channel at twice his speed. Now new area is being covered that will improve the searcher's chance of encountering the target.

2. Changes to the Simulation Model

In an attempt to further simplify the model and to make it's assumptions more closely in-line with those of theoretical searches, the concepts of the 6-ping history for determining cell coverage and the effective cell are dropped. This results in an omni-directional sensor that provides nearly continuous coverage of the search track. Because the sensor relies on discrete ping events, it is incorrect to assume continuous coverage. If the relative speed between the target and the searcher is such that the distance traveled between pings is less than the size of a ping cell, all cells will be covered and the sensor is effectively continuous.

Figure 5 shows one analysis period of cell coverage for a searcher using a continuous, omni-directional sensor and following the same search track as in Figure 2. Note that more of the cells in the partitioned area are covered using the continuous sensor than using the 6-ping history sensor of Figure 2. This effect is especially pronounced when the searcher starts moving, changes direction, or stops moving.

Appendix A contains a listing of the source code for the new classes that are used to implement the changes to the TDA. The simulation methodology remains the same as for the original analysis of the TDA and is implemented by the class Simulation2. First, a random search track is run and the ping cell matrix is updated to reflect the search effort. Second, the same search track is run against a number of uniformly penetrating targets and Pd is calculated. Third, the measures are calculated from the ping cell matrix information and the results are written to a file. Once all of the search tracks are run, the resulting measures and their corresponding Pd values are plotted and analyzed.

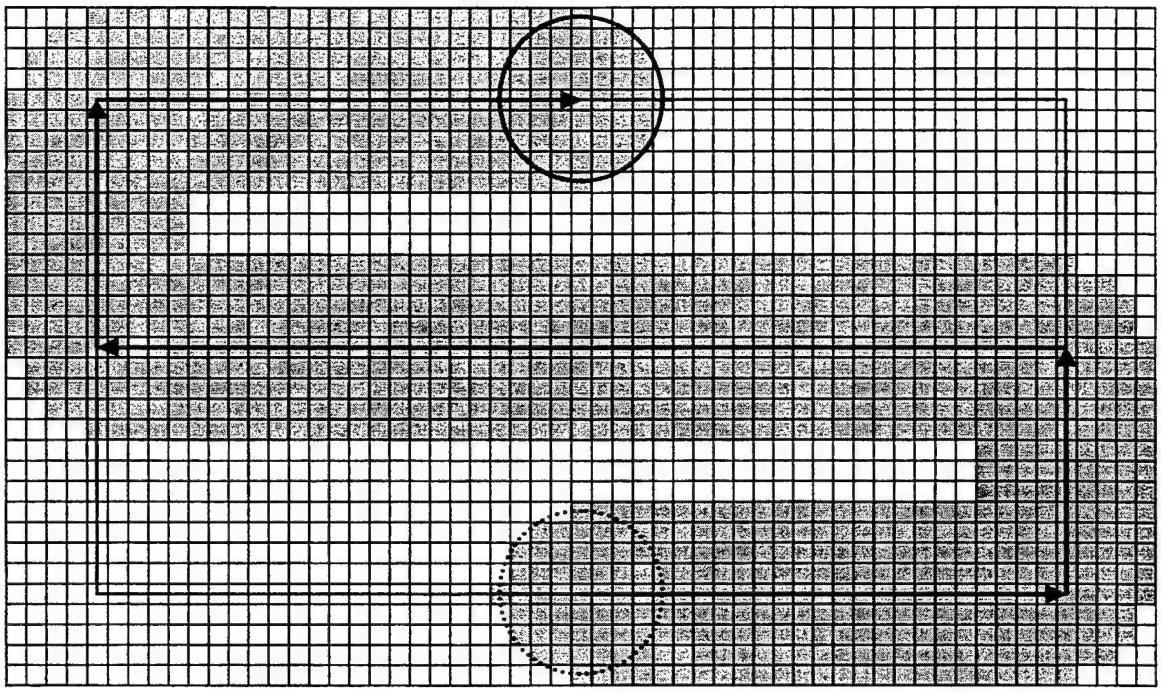

Figure 5. Cell Coverage Using a Continuous Omni-directional Sensor

The class PingCellMatrix1, described in Chapter II, partitioned the search area into a matrix of ping cells that contained information about the amount of search effort in geographic space. Transitioning to target-relative space requires a larger ping cell matrix because adjusting for the target's motion has the effect of making the barrier wider. The length of the barrier is the distance along which the target transits as shown in Figure 1. The width of the barrier remains constant because the target transits perpendicular to the width of the barrier. Because the target has no speed across the width of the barrier, the relative speed of the searcher is equal to his actual speed in that direction.

The following describes how the size of the new ping cell matrix is determined in target-relative space. As the target approaches to within one detection range of the top of the barrier, the geographic search area is all in front of him. As the target crosses the barrier, the search area ahead of him shrinks while the area behind him grows. The search area appears to slowly slip past the target until, at the end of the analysis period, the entire area is behind him. The result is that, from the target's perspective, the total apparent area available to the searcher is actually twice the geographic area.

Figure 6 shows the effects of modeling search effort in target-relative space on the search track of Figure 5. In geographic space, the searcher moves back-and-forth across the width of the barrier and moves from the bottom to the top of the barrier over the course of the track. In target-relative space, the searcher moves in a direction and with a speed equal to the sum of his geographic velocity vector and the velocity vector that is in the opposite direction of the target's.

Because the searcher is constrained to the search area, at some point during the barrier crossing the searcher will pass abeam of the target. As a consequence of this and the assumption of perfect sensors, there must be a non-zero Pd for every search track against the target.

The class PingCellMatrix2 has similar responsibilities as PingCellMatrix1, but the ping cell matrix it creates is based on the partitioning of a search area that is twice as wide as in geographic space. As discussed above, the search area appears to move past the target during his crossing. As a result, the center row of the target-relative ping cell matrix is the only place that a target can exist and all updates to the matrix must be relative to this row. This row is called the target row, shown as the middle row in Figure 6. Note that the cells in the border around the search area are not included in the target row. These cells are not part of the penetrated barrier width so a target cannot exist in them.

When a ping event occurs, the doPing() method of PingCellMatrix2 determines the geographic positions of both the target and the searcher from the simulation. It then calculates the row and column of the searcher within the ping cell matrix relative to the target row and updates the ping cell matrix using the masking matrix for the sensor.

B. RESULTS

After making the above changes to the model, the simulation ran under conditions similar to Simulation1 to determine the effect of modeling in target-relative space. Simulation2 uses Measures 1 and 2 of the TDA by calling the getMeasures() method of

the class Measures1. The results of this run are similar to those in Figures 1 and 2 – high variability and no strong relationship between the measure and Pd.

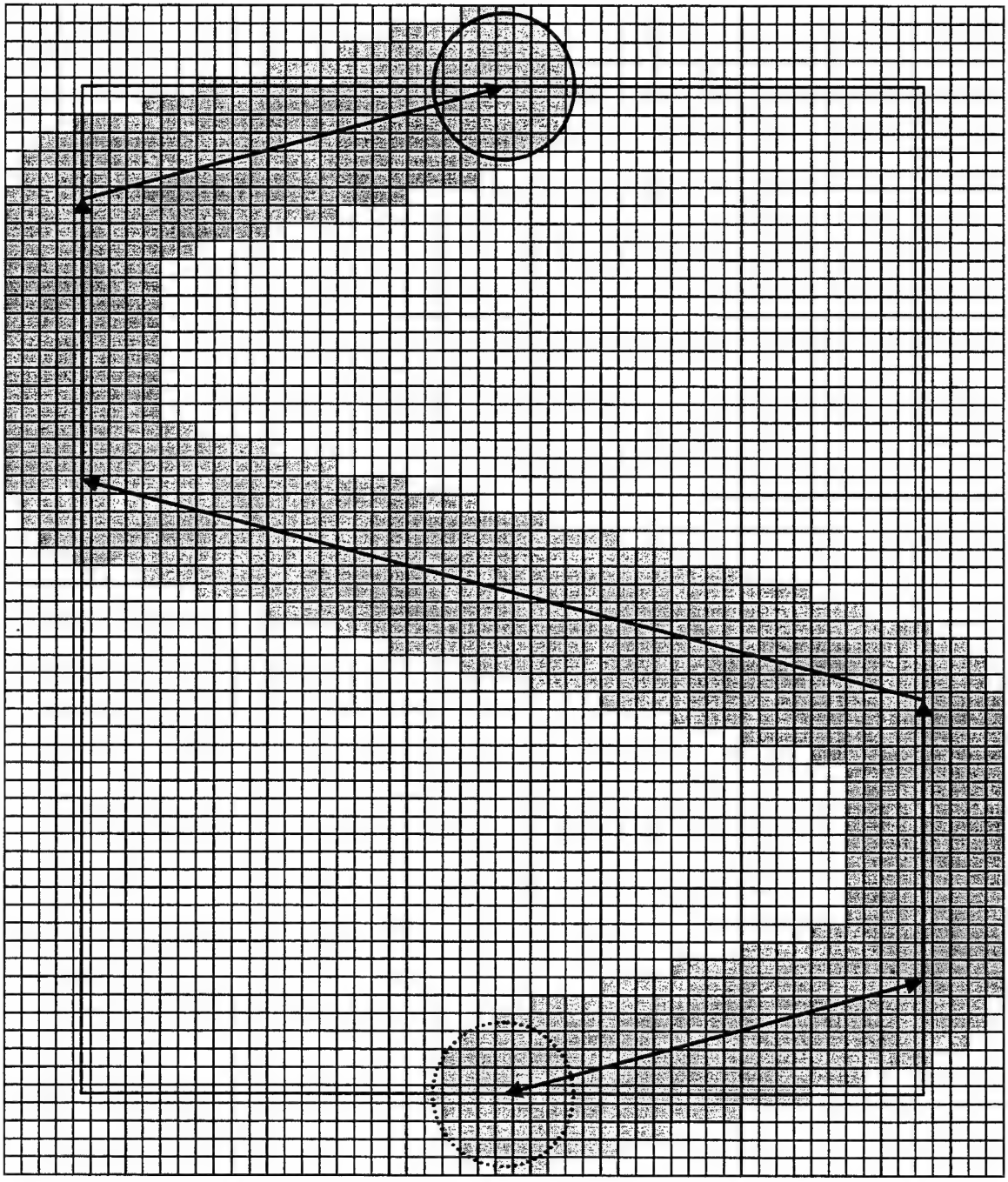

Figure 6. Cell Coverage in Target-relative Space

One possible explanation for the lack of improvement in the proposed measures is that both the TDA and the simulation use irrelevant portions of the search area in their calculations. Irrelevant, in this case, is a portion of the search area that does not contribute to Pd even though it is searched. In geographic space, it seems reasonable to use all of the available data in the ping cell matrix. In target-relative space, only the target row of the ping cell matrix is needed for calculations.

As discussed previously, at some point during the crossing, the searcher must pass abeam of the target. In fact, the searcher may pass abeam of the target more that once depending on the search track he chooses. It follows that the more time the searcher spends looking relative to the target in this way, the better the chance of detecting him.

Because the target eventually passes abeam of the searcher, determining the fraction of the target row that is covered by the searcher should provide an approximation to Pd. The class Measures2 does precisely this. When it's getMeasures() method is called, it scans the ping cells that are in the target row and returns the fraction that are covered.

Initial runs of Simulation2 using Measures2 provide hope that the new method of calculating the measure is an improvement. To produce more measure versus Pd data over the entire range of values, the simulation generates nine runs of 100 random search tracks. Each search track runs against 30 uniformly penetrating target tracks. The nine runs use all of the combinations of v=6, 12, and 18 kt searchers with a SW=4000, 8000, and 16000 yd sensor against a u=4 kt target in a 12 nm by L=24 nm area. Figure 7 is the compilation of the 900 data points from the runs. The least-squares regression line in Figure 7 suggests that there is a strong positive relationship between Measures2 and Pd.

The test scenario from Chapter II is run again using Measures2 to illustrate the significant difference between the two ways of calculating the measures. The result of this run is shown in Figure 8. Figure 9 is the result of the same test scenario run against 500 target tracks for each search track. This is done to obtain a more accurate Pd value. The worst case 95% large-sample confidence interval for the data in Figure 9 is Pd=0.5 $\pm$ 0.044.

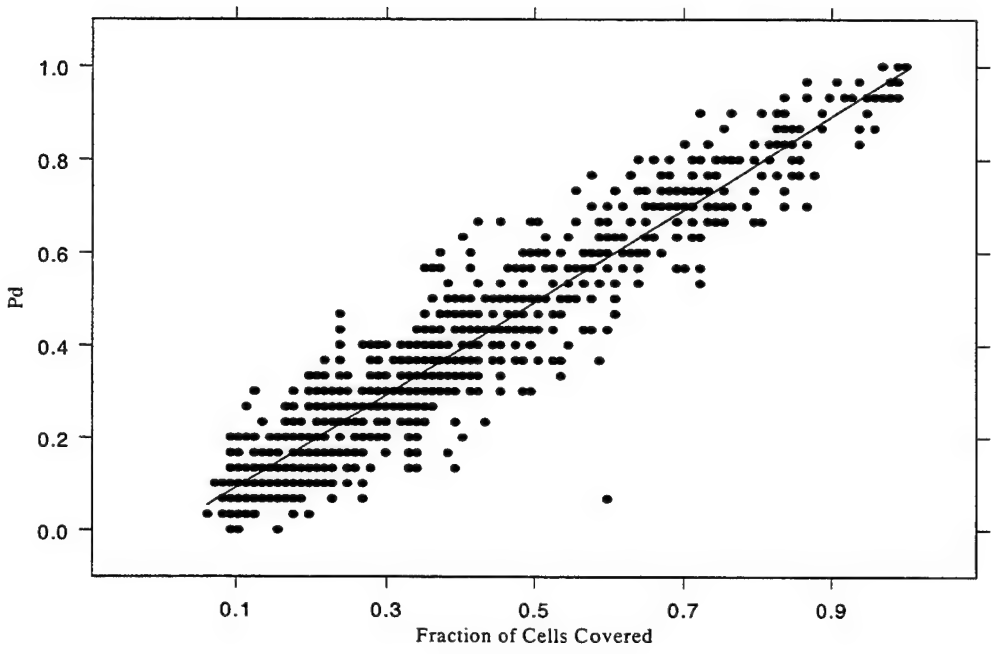

Figure 7. Measures2 – Fraction of Cells Covered in the Target Row

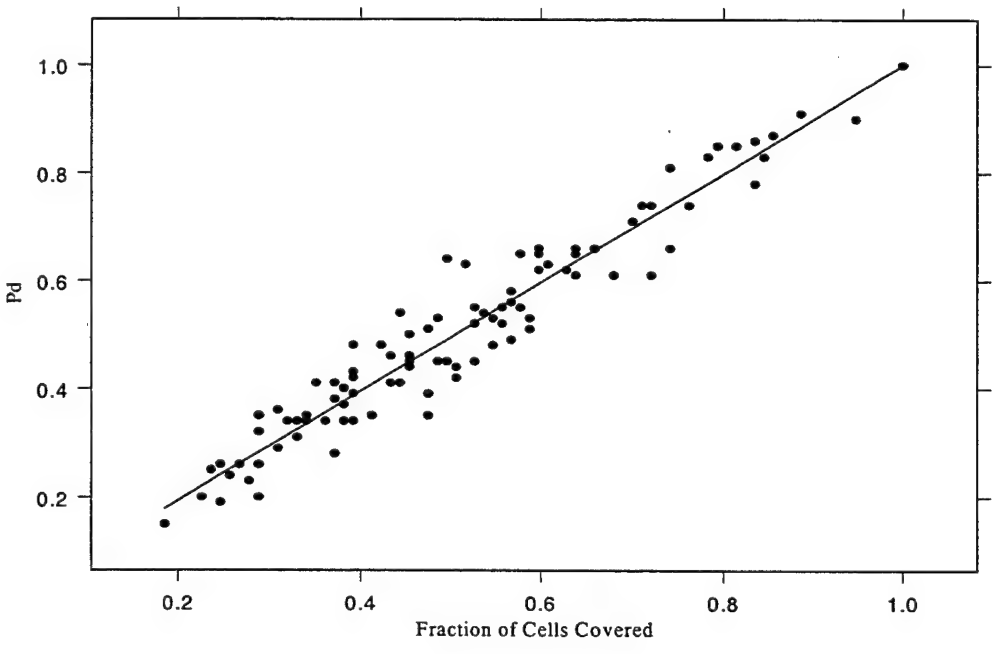

Figure 8. Test Scenario Using Measures2

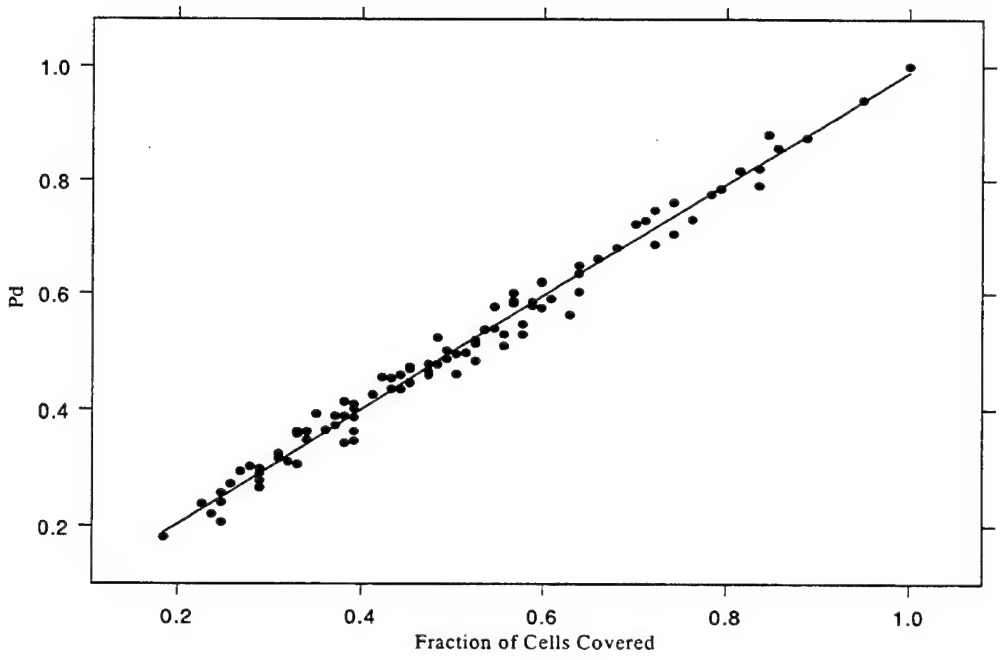

Figure 9. Test Scenario Using Measures2 Against 500 Target Tracks

The principle result of these simulation runs is that, by modeling in target-relative space, the barrier search Pd for an arbitrary search track can be calculated by determining the fraction of the discrete ping cells in the target row that are covered by the searcher as a target crosses the barrier.

Koopman [Ref. 1] introduced the idea that theoretical Pd is the ratio of the area actually searched to the total area to be searched for one half cycle of a "back-and-forth" or a "crossover" barrier search. Koopman's Pd formulas require a repeating pattern composed of straight legs. The pattern must repeat so that a half cycle may be defined and Pd calculated. The legs must be straight because the formulas are the analytic solutions to geometry problems that are not easily solved, if at all, for a curved path.

Because actual tracks from ASW barrier searches are neither straight, nor repeating, Koopman's formulas cannot be directly used. The results of this chapter suggest that by shrinking the area to be covered down to a row to be covered and by

calculating Pd at the end of an analysis period vice the end of a half cycle, barrier search Pd can be calculated for any arbitrary search track.

This chapter describes changes made to the simulation in an attempt to improve how well the proposed measures predict search effectiveness. Modeling the search in target-relative space is not, by itself, adequate for improving the measures. By also changing the method for calculating Measure 1, the simulation suggests that the method yields Pd of a single target crossing a barrier searched with an arbitrary search track.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. A GENERAL BARRIER SEARCH ALGORITHM

The simulation results of Chapter III suggest that it is possible to calculate Pd of a single target crossing a barrier searched with an arbitrary search track. The method used to obtain the results, however, suffers two drawbacks. First, the SIMKIT package used to generate and run the simulation is not commercially available and requires special training to understand and use. Second, Pd is calculated for only one target.

Both of these drawbacks are addressed in this chapter where "instantaneous Pd" is calculated as the fraction of covered cells in a single row of the ping cell matrix. Because information about the target, the searcher, and the search area is all that is required to perform this calculation, the need for simulation is eliminated and the other rows of the ping cells matrix can be used to calculate Pd for a continuous stream of targets

This chapter describes the development of a general barrier search algorithm that calculates instantaneous Pd for a continuous stream of targets that uniformly penetrate a barrier patrolled by a searcher following an arbitrary search track. The algorithm is then verified using a simple barrier search that has an analytic solution. In the final part of the chapter, the generality of the algorithm is demonstrated by using it to evaluate an arbitrary search track.

A. ALGORITHM DEVELOPMENT

The general barrier search algorithm is implemented in Java™, but can be implemented in other programming languages. Appendix B contains a listing of the pseudo-code for the algorithm. The algorithm reads information about the search directly from a file to eliminate the need to recompile the program each time the search is changed. Search parameters and search track data are contained in the file. This information initializes the variables used by the algorithm. Note that the algorithm does not check the track data to ensure it is within the search area. Next the ping cell matrix and the sensor's masking matrix are created and initialized. The masking matrix is the same as previously discussed.

Two types of ping cell matrices have been discussed so far – geographic and target-relative. The information in a single row of the target-relative ping cell matrix of Chapter III is used to calculate Pd for a single, uniformly distributed target crossing the barrier. The target-relative ping cell matrix is twice as large as the geographic ping cell matrix so that search effort is modeled with respect to a stationary target. The general barrier search algorithm uses a third type of ping cell matrix.

The new ping cell matrix is the same size, and the ping cells are updated in the same way, as the geographic matrix. This matrix layout physically represents the search area, unlike the target-relative matrix. The new matrix is dynamically updated. This is done to physically model the motion of a continuous stream of uniformly distributed targets.

To model the target escaping from the bottom of the barrier, Pd is calculated using the cell coverage information in the bottom row of the matrix. Then, to model the motion of the targets currently crossing the barrier, the information in each row of the matrix is moved down one row closer to the bottom. This leaves the top row of the matrix empty, representing a undetected target entering the top of the barrier. *When this procedure is followed, the fraction of cells in the bottom row that are searched is the probability of detecting a target, that is uniformly distributed across the width of the barrier, that started at the top of the barrier one analysis period earlier.*

Figures 10 though 13 show a graphical representation of the state of the ping cell matrix for a searcher performing a back-and-forth barrier search. This example is for a 5 kt target crossing a barrier that is partitioned using a 500 yd cell size. Given those parameters, it takes a target 3 minutes to move from one row of cells to the next. This means that every three minutes, a Pd value is calculated and the ping cell matrix is updated as described above. If an active sonar ping occurs in the time between updates to the ping cell matrix, then the ping cells are updated as for a geographic ping cell matrix.

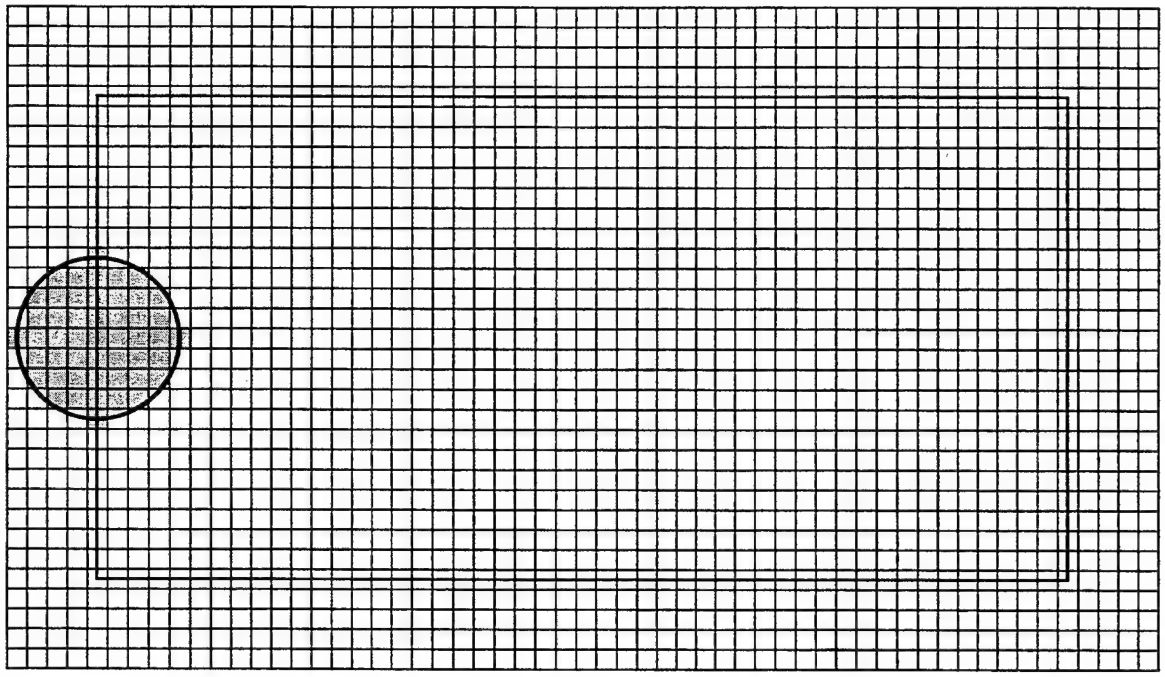

Figure 10. Initial State of the Ping Cell Matrix at Minutes

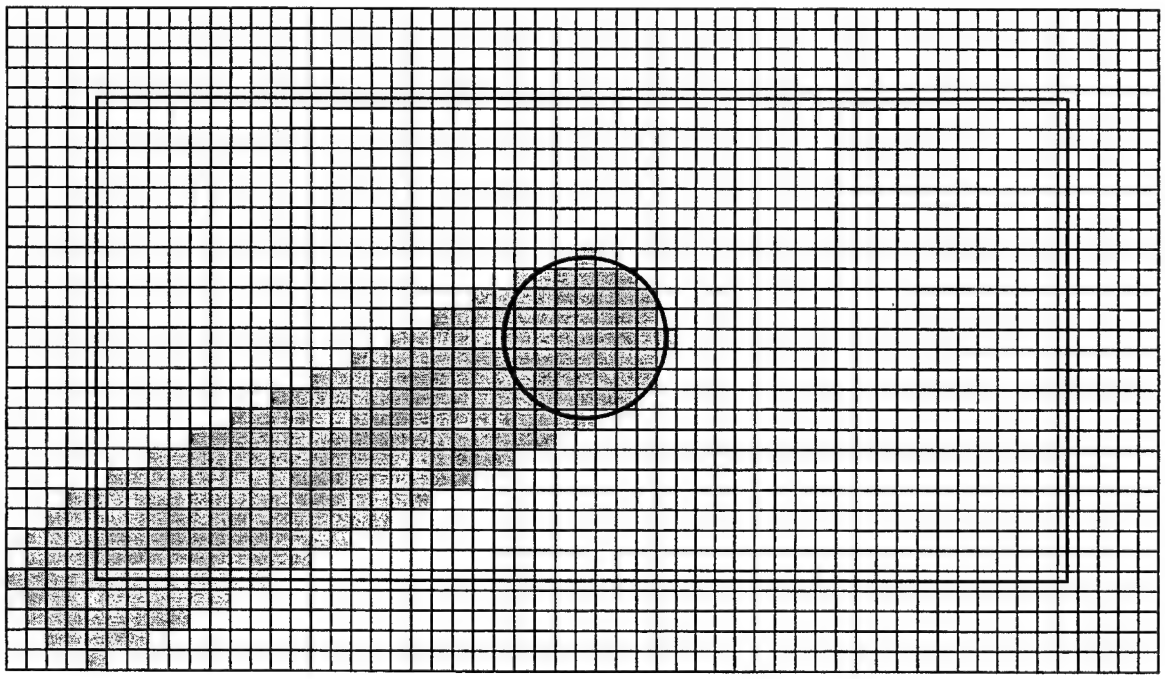

Figure 11. State of the Ping Cell Matrix at 36 Minutes

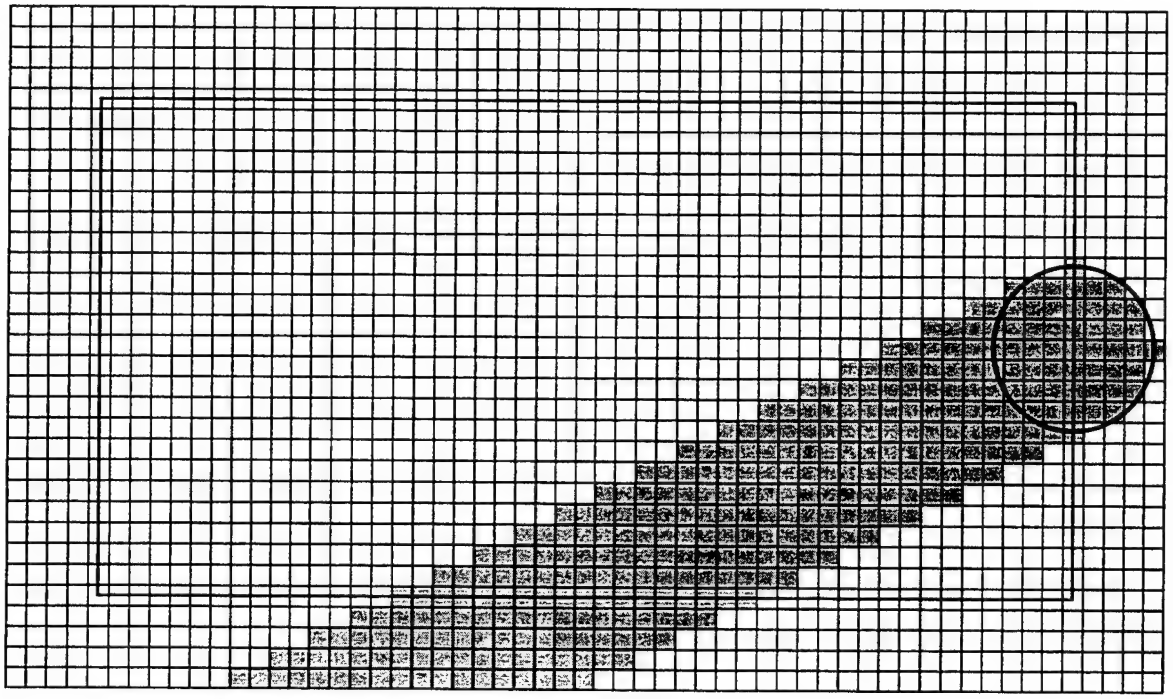

Figure 12. State of the Ping Cell Matrix at 72 Minutes

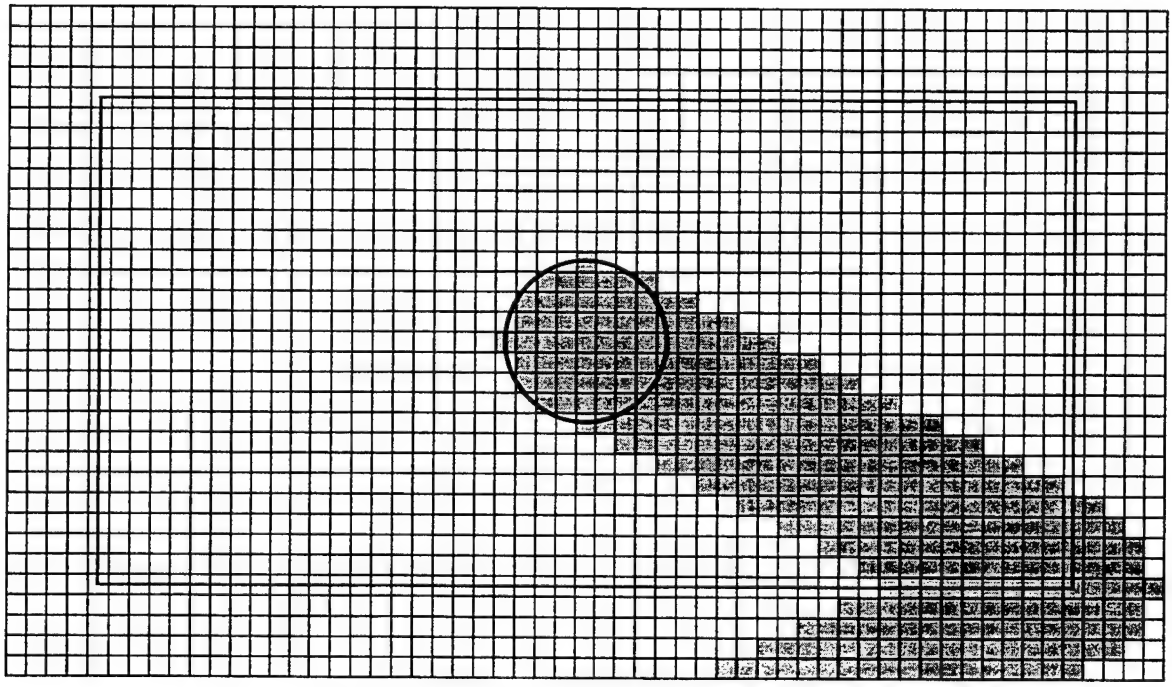

Figure 13. State of the Ping Cell Matrix at 108 Minutes

Figure 10 shows the first ping that occurs when the searcher is on the left edge of the search area. The first ping detects all target cells within range. The searcher continues to ping as he moves horizontally to the right across the barrier. Every three minutes, all targets move one row down the matrix and eventually reach the bottom. Then, the fraction of cells detected in the bottom row is calculated. This is the instantaneous Pd and conceptually is the Pd of a target, uniformly distributed across the top of the barrier, that started moving 99 minutes earlier.

As shown in Figure 11, during the first 36 minutes, none of the cells in the bottom row of the matrix are covered, resulting in Pd=0. By the time the searcher reaches the right edge of the searcher area, Pd values have reached a constant value. For any row in the ping cell matrix, 49 cells comprise the width of the barrier and eight cells comprise the border. From Figure 12, the next target escaping the barrier has a Pd=0.37 because 18 of the 49 cells in the bottom row are covered by the searcher.

The ping cell matrix in this example has 33 rows. A uniformly distributed target starting at the top of the barrier will require 99 minutes to cross the barrier at 3 minutes per row. This is equal to the analysis period of Chapters II and III. Pd values are only valid after the first analysis period has passed since the start of the algorithm. This is because Pd values that are calculated before this are for targets that did not cross the entire barrier, resulting in an underestimation of Pd.

B. OUTPUT

1. For a Back-and-forth Barrier Search

Initial verification and validation of the algorithm uses search data from a back-and-forth barrier search. Because the back-and-forth barrier search is a repeating pattern, the Pd values should approach steady state. Figure 14 shows the output of the algorithm for a v=12 kt searcher with a SW=4000 yd sensor against a u=8 kt target in a 10 nm by L=14 nm area. The search track for this run is for a searcher that comes no closer than

one detection range from the left and right edges of the barrier to match the conditions of equation 1.3-3 of [Ref. 2] which results in Pd=0.258.

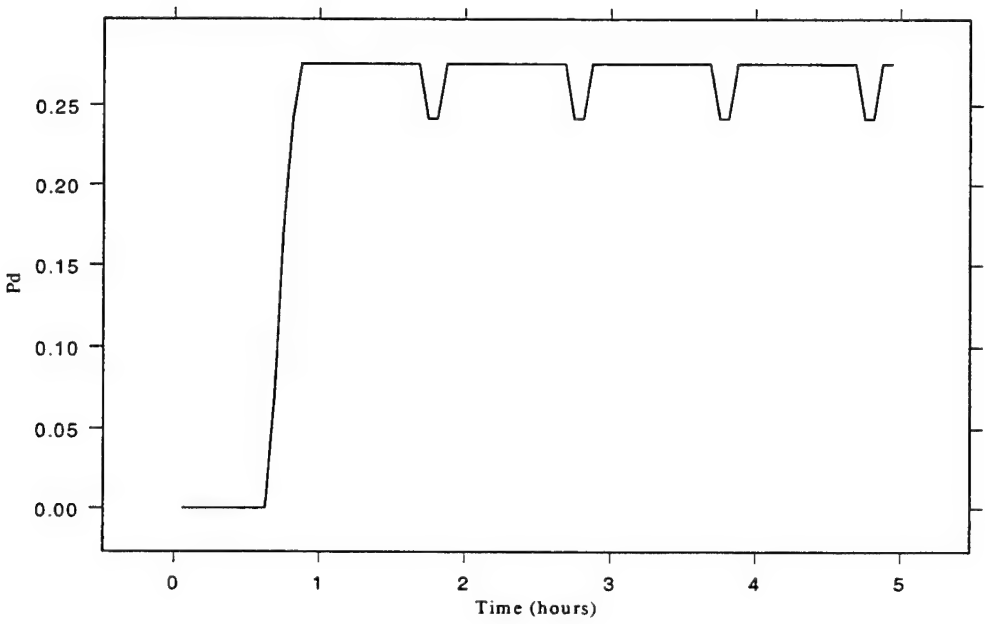

Figure 14. Algorithm Output for a Back-and-forth Barrier Search

Pd values are initially zero because the searcher in this scenario is in the middle of the barrier, but they quickly rise to a value of Pd=0.276. Pd values do not reach a steady-state value as expected. With a period of one hour, they dip down to Pd=0.241. This does not occur by coincidence, but as a result of the searcher reaching the edge of the barrier and reversing course. A close inspection of Figure 13 reveals that the fraction of the barrier width covered does decrease in the vicinity of the barrier edge because of the circular shape of the sensor. Since that fraction is how the algorithm calculates Pd, the effect is seen in the output. For this scenario the difference between the steady-state value and the dip is the difference between 8 cells and 7 cells out of 29 being covered in the target rows.

Koopman's formulas [Ref. 1] require a repeating search track with straight legs. Because his formulas are calculated for half of the repeating cycle, they effectively average out the Pd dips that occur when the values are calculated instantaneously. For a

repeating track, even one without straight legs, the output of the algorithm is periodic after the first analysis period. This is the case in Figure 14. If a sliding average is calculated with an averaging period equal to the period of the output data, the average will eventually reach a constant value. Because most search tracks that are analyzed with the algorithm do not repeat, the output is not averaged, but left as discrete instantaneous values.

2. For a Random Search

The final step in verifying the algorithm is to analyze an arbitrary search track and generate a graph similar to Figure 14. Figure 15 shows the search track that is used to test the algorithm. The search track is for a v=12 kt searcher with a SW=4000 yd sensor in a 10 nm by L=14 nm area. The searcher starts at the center of the area and searches for five hours. The axis for Figure 15 are the coordinates of the searcher (in nm) with the origin being the center of the area.

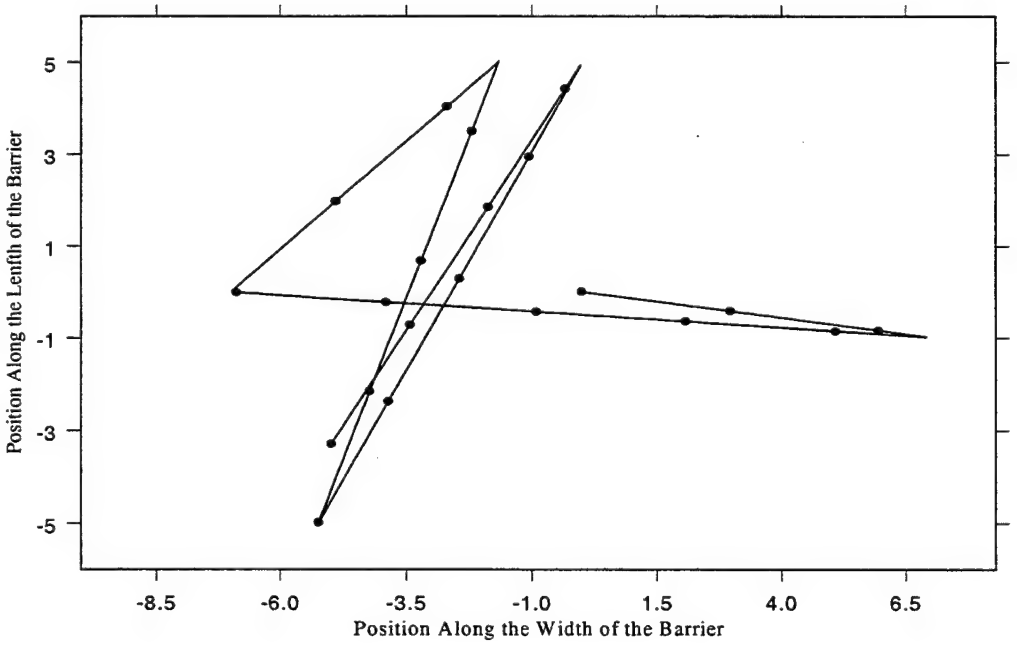

Figure 15. Search Track Data for a Random Barrier Search

Because the relative speed between the searcher and target affects cell coverage and ultimately the Pd values, each graph of instantaneous Pd is only valid for the assumed target speed. Figure 16 shows the result of running the search track of Figure 15 against a 6 kt target. To aid in the analysis of the output, the position of the searcher along the length of the barrier (the searcher's y-coordinate) versus time is also plotted on Figure 16.

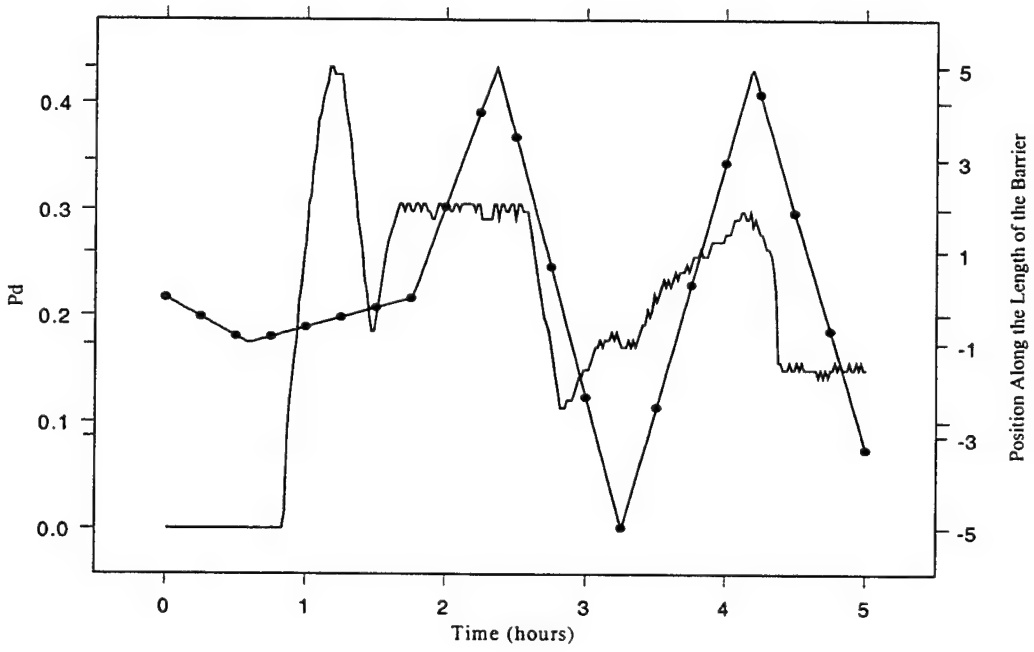

Figure 16. Algorithm Output for a Random Barrier Search

As seen in Figure 16, there is an initial transient period where the Pd starts at zero and increases to a certain value. Three distinct Pd periods are defined and explained.

- 1:45 to 2:30

  From the start of the search until 1:45, the searcher moves primarily back-and-forth across the width of the barrier with very little motion along the length of the barrier. After the initial transient period, this back-and-forth search produces a period of stable Pd values. At 1:45 the searcher moves toward the

top of the barrier. Because the cells at the bottom of the ping cell matrix are already filled and take some time to exit the barrier, Pd continues to be stable until 2:30 when the effect of the increased relative motion between the searcher and the target is realized, resulting in a dip in Pd.

- 3:00 to 4:30

  Between 2:15 and 2:30, the searcher reaches the top of the area and must turn around. He begins a long move that takes him to the bottom of the barrier at roughly twice the speed of the target, resulting in a gradual increase in Pd. At 3:15, the searcher reaches the bottom of the barrier and must turn around. Once again, there is an increase in the relative motion between the searcher and the target which causes the Pd dip from 4:15 to 4:30.

- 4:30 to 5:00

  Because the searcher moved from the bottom to the top of the search area at a constant speed from 3:15 to just before 4:15, all of the target rows that exit the ping cell matrix starting at 4:30 are covered equally, resulting in stable Pd values. The effect of the next leg of downward motion by the searcher is not displayed.

C. USES AND FURTHER DEVELOPMENT

1. An Evaluative TDA for SHAREM Barrier Events

As stated in the Chapter II, SHAREM exercises are divided into planning, execution, and analysis phases. The general barrier search algorithm that has been developed can be incorporated into an evaluative TDA that can be used in all phases of SHAREM exercises.

During the planning phase, different search patterns can be tested against the threat parameters to determine general guidelines for conducting the search. If the search patterns are found to be inadequate due to short detection ranges, then more search assets can be incorporated into the search plan.

During the execution phase, real-time graphs such as Figures 15 and 16 can be generated to asses how well the search was executed over short periods like the span of one watch. A post-watch analysis of the graphs would provide immediate feedback to the watch team on how their actions affect the likelihood of success. For example, if a ship were required to conduct helicopter launch and recovery operations during the search, the affect of the restrictions on the ship's movement can be readily seen.

During the analysis phase, the execution of the barrier search event is evaluated as a whole. By reviewing the search tracks, predicted Pd values, and actual detections over the course of a 24 hour event, tactics can be identified that produce more effective searches. Additionally, an overall quantitative assessment can be made of the performance of the search assets.

2. An Expandable Algorithm

Admittedly, the general barrier search algorithm is very simple and makes a number of assumptions that are not realistic. These assumptions are used to produce results that can be verified and validated using analytic solutions from search theory. It does, however, provide a base from which to expand. Additional measures can be added to determine how much search effort is wasted by searching outside of the assigned area, the sensor coverage can be modified by changing the values of the masking matrix, detection rate models can be used instead of perfect sensors, and search areas can be combined to calculate measure for the search force. Any or all of these improvements can be added based on the desired use of the algorithm.

V. CONCLUSIONS

The first part of this research reviews two of several new proposed measures that are part of a TDA used to quantify the execution of ASW barrier searches with active sonar. Discrete event simulation is used to determine barrier search Pd and calculate the proposed measures. When the measures are compared to Pd, they do not appear to adequately predict effectiveness for the most general search.

In an attempt to improve the measures, the simulation model is changed to account for the relative motion between the target and the searcher. By modeling the search effort in target-relative space and calculating the correct fraction of the search area that is covered by the searcher, a new measure results which is the Pd of a single target crossing the barrier.

The second part of this research extended the new method for calculating Pd so that it can evaluate a search track of any duration without using simulation. The resulting algorithm determines Pd for a continuous stream of transiting targets as they escaped from the bottom of the barrier. The algorithm's output data is plotted versus time to provide a graphical representation of the effect that a searcher's track has on Pd.

Although the algorithm is quite simple, it can easily be expanded to model more complex sonar systems, to calculate Pd for multiple searchers, and to calculate other measures that are of interest in analyzing ASW barrier searches.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SOURCE CODE FOR THE SWDG PACKAGE

```
package swdg;

import java.io.*;
import simkit.*;
import simkit.smd.*;
import simkit.data.*;

public class Simulation {

     public static FileWriter simOutput;

     public static void main(String[] args) {
          try {
               simOutput=new FileWriter("sim1run8.txt");
          } catch(IOException e) {
               System.err.println(e);
          }

          //area dimensions are nautical miles
          double[] area=new double[4];
          area[0]=12.0;
          area[1]=0;
          area[2]=24.0;
          area[3]=0;

          //speed is knots, range is yards
          double searcherSpeed=12.0;
          double detectionRange=8000.0;

          //speed is knots
          double targetSpeed=4.0;

          //cell size is yards, ping interval is hours
          double cellSize=500.0;
          double pingInterval=1.0/60.0;

          int searchTracks=100;
          int targetTracks=500;

          MediatorFactory.addMediatorType("simkit.smd.
          BasicSensor", "swdg.ResetBasicMover",
          simkit.smd.CookieCutterMediator");
```

```
Mover searcher=new ResetBasicMover("SEARCHER",
new Coordinate(), searcherSpeed);

Referee.DEFAULT_REFEREE.registerTarget(searcher);

RandomMoverManager3 forSearcher=new
RandomMoverManager3(searcher, area);

Sensor sonar=new BasicSensor(searcher,
detectionRange/2000);

Referee.DEFAULT_REFEREE.registerSensor(sonar);

RandomSensorManager forSonar=new
RandomSensorManager(sonar);

Mover target=new ResetBasicMover("TARGET", new
Coordinate(), targetSpeed);

TargetMoverManager forTarget=new
TargetMoverManager(target, area,
detectionRange/2000);

PingCellMatrix1 pcm=new PingCellMatrix1(area,
detectionRange, cellSize, pingInterval,
searcher);

for(int i=0; i<searchTracks; i++) {
     Referee.DEFAULT_REFEREE.unRegisterTarget
     (target);
     forSearcher.setSeed(1987911+i);
     Schedule.reset();
     pcm.waitDelay("Ping", 0.0);
     Schedule.startSimulation();

     double[] output=Measures.getMeasures(pcm);
          for(int index=0; index<output.length;
          index++) {
               fileWrite(Double.toString
               (output[index])+"\t");
          }

     Referee.DEFAULT_REFEREE.registerTarget
     (target);
```

```
                for(int j=0; j<targetTracks; j++) {
                    forSearcher.setSeed(1987911+i);
                    Schedule.reset();
                    Schedule.startSimulation();
                }
                fileWrite(Double.toString(forSonar.
                getNumberDetections()/(double)targetTracks)+
                "\n");
                forSonar.resetNumberDetections();
            }

            try {
                simOutput.close();
            } catch(IOException e) {
                System.err.println(e);
            }
        }

        public static void fileWrite(String data) {
            try {
                simOutput.write(data);
            } catch(IOException e) {
                System.err.println(e);
            }
        }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
package swdg;

import java.io.*;
import simkit.*;
import simkit.smd.*;
import simkit.data.*;

public class RandomMoverManager3 extends SimEntityBase {

     private Mover moverReference;
     private double theta;
     private double[] area;
     private boolean top;
     private boolean bottom;
     private boolean right;
     private boolean left;
     private RandomNumber uniform;

     public RandomMoverManager3(Mover moverReference,
     double[] area) {
          this.moverReference=moverReference;
          moverReference.addSimEventListener(this);
          this.area=area;
          uniform=RandomFactory.getRandomNumber();
     }

     public void doRun() {
          double x=area[3]+(area[2]-area[3])*
          uniform.draw();

          double y=area[1]+(area[0]-area[1])*
          uniform.draw();

          ((ResetBasicMover)moverReference)
          .setOriginalLocation(x,y);

          theta=2*Math.PI*uniform.draw();

          moverReference.moveTo(findBarrierCrossing
          (theta));
     }

     public void reset() {
          top=false;
          bottom=false;
          right=false;
```

```
        left=false;
    }

    public void doEndMove(Mover randomMover) {
        if(top) {
            theta=generateCosine(uniform.draw())+
            1.5*Math.PI;

            top=false;
        }
        else if(bottom) {
            theta=generateCosine(uniform.draw())+
            0.5*Math.PI;

            bottom=false;
        }
        else if(right) {
            theta=generateCosine(uniform.draw())+
            Math.PI;

            right=false;
        }
        else if(left) {
            theta=generateCosine(uniform.draw());
            left=false;
        }
        moverReference.moveTo(findBarrierCrossing
        (theta));
    }

    public Coordinate findBarrierCrossing(double theta) {
        double xo=moverReference.getCurrentLocation()
        .getXCoord();

        double yo=moverReference.getCurrentLocation()
        .getYCoord();

        if(theta>0.0 && theta<0.5*Math.PI) {
            double yf=(area[2]-xo)*Math.tan(theta)+yo;
            if(yf<area[0]) {
                right=true;
                return(new Coordinate(area[2], yf));
            }
            else if(yf>area[0]) {
                top=true;
                double xf=(area[0]-yo)/Math.tan(theta)
```

```
                +xo;

                return(new Coordinate(xf, area[0]));
            }
            else {
                return(new Coordinate(area[2],
                area[0]));
            }
        }
        else if(theta>0.5*Math.PI && theta<1.0*Math.PI) {
            double yf=(area[3]-xo)*Math.tan(theta)+yo;
            if(yf<area[0]) {
                left=true;
                return(new Coordinate(area[3], yf));
            }
            else if(yf>area[0]) {
                top=true;
                double xf=(area[0]-yo)/Math.tan(theta)
                +xo;

                return(new Coordinate(xf, area[0]));
            }
            else {
                return(new Coordinate(area[3],
                area[0]));
            }
        }
        else if(theta>1.0*Math.PI && theta<1.5*Math.PI) {
            double yf=(area[3]-xo)*Math.tan(theta)+yo;
            if(yf>area[1]) {
                left=true;
                return(new Coordinate(area[3], yf));
            }
            else if(yf<area[0]) {
                bottom=true;
                double xf=(area[1]-yo)/Math.tan(theta)
                +xo;

                return(new Coordinate(xf, area[1]));
            }
            else {
                return(new Coordinate(area[3],
                area[1]));
            }
        }
        else if(theta>1.5*Math.PI && theta<2.0*Math.PI ||
```

```
            theta>-0.5*Math.PI && theta<0.0) {
                double yf=(area[2]-xo)*Math.tan(theta)+yo;
                if(yf>area[1]) {
                    right=true;
                    return(new Coordinate(area[2], yf));
                }
                else if(yf<area[1]) {
                    bottom=true;
                    double xf=(area[1]-yo)/Math.tan(theta)
                    +xo;

                    return(new Coordinate(xf, area[1]));
                }
                else {
                    return(new Coordinate(area[2],
                    area[1]));
                }
            }
            else {
                System.out.println("DIRECTION");
                System.exit(0);
            }
            return(new Coordinate());
        }

        public double generateCosine(double w) {
            return(Math.asin(2*w-1));
        }

        public void setSeed(long seed) {
            uniform.setSeed(seed);
        }
}
```

```
package swdg;

import simkit.*;
import simkit.smd.*;
import simkit.data.*;

public class TargetMoverManager extends SimEntityBase {

     //instance variables
     private Mover moverReference;
     private double[] area;
     private double detectionRange;
     private RandomNumber uniform;

     //constructor methods
     public TargetMoverManager(Mover moverReference,
     double[] area, double detectionRange) {
          this.moverReference=moverReference;
          moverReference.addSimEventListener(this);
          this.area=area;
          this.detectionRange=detectionRange;
          uniform=RandomFactory.getRandomNumber();
     }

     //instance methods
     public void doRun() {
          double x=area[3]+(area[2]-area[3])*
          uniform.draw();

          ((ResetBasicMover)moverReference)
          .setOriginalLocation(x, area[0]+detectionRange);

          moverReference.moveTo(new Coordinate(x, area[1]-
          detectionRange));
     }

     public void doEndMove(Mover targetMover) {
          Schedule.stopSimulation();
     }

     public void setSeed(long seed) {
          uniform.setSeed(seed);
     }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
package swdg;

import simkit.*;
import simkit.smd.*;

public class PingCellMatrix1 extends SimEntityBase
implements PingCellMatrix {

     private double[] area;
     private double detectionRange;
     private double cellSize;
     private double pingInterval;
     private Mover moverReference;

     private SixPingCell[][] cell;
     private boolean[][] mask;

     private int midRow;
     private int midColumn;

     public PingCellMatrix1(double[] area, double
     detectionRange, double cellSize, double pingInterval,
     Mover moverReference) {
          this.area=area;
          this.detectionRange=detectionRange;
          this.cellSize=cellSize;
          this.pingInterval=pingInterval;
          this.moverReference=moverReference;

          createCellMatrix();
          createMaskingMatrix();
     }

     public double getDetectionRange() {
          return(detectionRange);
     }

     public double getCellSize() {
          return(cellSize);
     }

     public double getPingInterval() {
          return(pingInterval);
     }

     public double getCurrentPingTime() {
```

```
        return(Schedule.simTime());
    }

    public PingCell[][] getCell() {
        return cell;
    }

    public void createCellMatrix() {
        int rows=(int)(Math.rint(((area[0]-
        area[1])*2000.0+2*detectionRange)/cellSize)+1);

        int columns=(int)(Math.rint(((area[2]-
        area[3])*2000.0+2*detectionRange)/cellSize)+1);

        cell=new SixPingCell[rows][columns];
        for(int i=0; i<cell.length; i++) {
            for(int j=0; j<cell[i].length; j++) {
                cell[i][j]=new SixPingCell(this);
            }
        }
    }

    public void createMaskingMatrix() {
        int rows=
        (int)(Math.rint(2*detectionRange/cellSize)+1);

        int columns=
        (int)(Math.rint(2*detectionRange/cellSize)+1);

        mask=new boolean[rows][columns];
        midRow=(mask.length-1)/2;
        midColumn=(mask[0].length-1)/2;
        for(int i=0; i<mask.length; i++) {
            for(int j=0; j<mask[i].length; j++) {
                double distance=Math.sqrt(Math
                .pow((i-midRow),2)
                +Math.pow((j-midColumn),2));

                if(distance<=detectionRange/cellSize) {
                    mask[i][j]=true;
                }
                else {
                    mask[i][j]=false;
                }
            }
        }
```

```
    }

    public void reset() {
        for(int i=0; i<cell.length; i++) {
            for(int j=0; j<cell[i].length; j++) {
                cell[i][j].reset();
            }
        }
    }

    public void doPing() {
        double x=
        moverReference.getCurrentLocation().getXCoord();

        double y=
        moverReference.getCurrentLocation().getYCoord();

        int yRow=(int)(Math.rint(((y-area[1])*2000.0
        +detectionRange)/cellSize));

        int xColumn=(int)(Math.rint(((x-area[3])*2000.0
        +detectionRange)/cellSize));

        for(int i=0; i<mask.length; i++) {
            for(int j=0; j<mask[i].length; j++) {
                if(mask[i][j]) {
                    cell[yRow-midRow+i][xColumn-
                    midColumn+j].incrementPings();
                }
            }
        }

        this.waitDelay("Ping", pingInterval);
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
package swdg;

public class Measures {

     public static double[] getMeasures(PingCellMatrix pcm)
     {
          double[] measures=new double[2];
          int notEffectiveIn=0;
          int effectiveIn=0;
          int notEffectiveOut=0;
          int effectiveOut=0;
          boolean out=false;

          PingCell[][] cell=pcm.getCell();
          int cellsOut=
          (int)Math.rint(pcm.getDetectionRange()/pcm.
          getCellSize());

          int rowInMin=cellsOut;
          int rowInMax=cell.length-1-cellsOut;
          int colInMin=cellsOut;
          int colInMax=cell[0].length-1-cellsOut;

          double moeOneNumerator=0;
          double stationArea=
          (double)cell.length*cell[0].length;

          for(int i=0; i<cell.length; i++) {
               for(int j=0; j<cell[i].length; j++) {
                    if(i<rowInMin || i>rowInMax ||
                    j<colInMin || j>colInMax) {
                         out=true;
                    }
                    else {
                         out=false;
                    }

                    int timesCovered=
                    cell[i][j].getTimesCovered();

                    moeOneNumerator=moeOneNumerator+(1-
                    Math.pow(Math.E, -timesCovered));

                    if(out) {
                         if(timesCovered==1) {
                              notEffectiveOut++;
```

```
                    }
                    else if(timesCovered>=2) {
                        effectiveOut++;
                    }
                }
                else {
                    if(timesCovered==1) {
                        notEffectiveIn++;
                    }
                    else if(timesCovered>=2) {
                        effectiveIn++;
                    }
                }
            }
        }
        measures[0]=moeOneNumerator/stationArea;
        measures[1]=(double)effectiveIn/stationArea;
        return(measures);
    }
}
```

```
package swdg;

public class SixPingCell implements PingCell {

     private PingCellMatrix pcmReference;
     private int pings;
     private double lastPingTime;
     private int timesCovered;

     public SixPingCell(PingCellMatrix pcmReference) {
          this.pcmReference=pcmReference;
          reset();
     }

     public int getPings() {
          return(pings);
     }

     public int getTimesCovered() {
          return(timesCovered);
     }

     public void reset() {
          pings=0;
          lastPingTime=-pcmReference.getPingInterval();
          timesCovered=0;
     }

     public void incrementPings() {
          if(Math.abs(pcmReference.getCurrentPingTime()-
          lastPingTime-pcmReference.getPingInterval())
          <0.001) {
               pings++;
               if(pings==6) {
                    pings=0;
                    timesCovered++;
               }
          }
          else {
               pings=1;
          }
          lastPingTime=pcmReference.getCurrentPingTime();
     }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
package swdg;

import java.io.*;
import simkit.*;
import simkit.smd.*;
import simkit.data.*;

public class Simulation2 {

     public static FileWriter simOutput;

     public static void main(String[] args) {
          double confidenceInterval;
          SimpleStats probabilityDetection=new SimpleStats
          (SamplingType.TALLY);
          try {
               simOutput=new FileWriter("sim2test.txt");
          } catch(IOException e) {
               System.err.println(e);
          }

          //area dimensions are nautical miles
          double[] area=new double[4];
          area[0]=12.0;
          area[1]=0;
          area[2]=24.0;
          area[3]=0;

          //speed is knots, range is yards
          double searcherSpeed=18.0;
          double detectionRange=4000.0;

          //speed is knots
          double targetSpeed=4.0;

          //cell size is yards, ping interval is hours
          double cellSize=500.0;
          double pingInterval=1.0/60.0;

          int searchTracks=100;
          int targetTracks=500;

          MediatorFactory.addMediatorType("simkit.smd
          .BasicSensor","swdg.ResetBasicMover",
          "simkit.smd.CookieCutterMediator");
```

```
Mover searcher=new ResetBasicMover("SEARCHER",
new Coordinate(),searcherSpeed);

RandomMoverManager3 forSearcher=new
RandomMoverManager3(searcher, area);

Sensor sonar=new BasicSensor(searcher,
detectionRange/2000);

Referee.DEFAULT_REFEREE.registerSensor(sonar);

RandomSensorManager forSonar=new
RandomSensorManager(sonar);

Mover target=new ResetBasicMover("TARGET", new
Coordinate(), targetSpeed);

TargetMoverManager forTarget=new
TargetMoverManager(target,area,
detectionRange/2000);

PingCellMatrix2 pcm=new PingCellMatrix2(area,
detectionRange, cellSize, pingInterval, searcher,
target);

for(int i=0; i<searchTracks; i++) {
     System.out.println("Calculating measures for
     search track "+i);

     Referee.DEFAULT_REFEREE.unRegisterTarget
     (target);

     pcm.setResetEnabled(true);
     forSearcher.setSeed(1987911+i);

     Schedule.reset();
     pcm.waitDelay("Ping", 0.0);
     Schedule.startSimulation();

     double[] output=Measures2.getMeasures(pcm);
     for(int index=0; index<output.length;
     index++) {

          fileWrite(Double.toString
          (output[index])+"\t");
     }
```

```
                System.out.println("Calculating Pd for
                search track "+i);

                Referee.DEFAULT_REFEREE.registerTarget
                (target);

                pcm.setResetEnabled(false);

                for(int j=0; j<targetTracks; j++) {
                     forSearcher.setSeed(1987911+i);
                     Schedule.reset();
                     Schedule.startSimulation();
                }
                fileWrite(Double.toString
                (forSonar.getNumberDetections()/(double)
                targetTracks)+"\t\n");

                forSonar.resetNumberDetections();
           }

           try {
                simOutput.close();
           } catch(IOException e) {
                System.err.println(e);
           }
      }

      public static void fileWrite(String data) {
           try {
                simOutput.write(data);
           } catch(IOException e) {
                System.err.println(e);
           }
      }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

```
package swdg;

import simkit.*;
import simkit.smd.*;

public class PingCellMatrix2 extends SimEntityBase
implements PingCellMatrix {

     private double[] area;
     private double detectionRange;
     private double cellSize;
     private double pingInterval;
     private Mover searcherReference;
     private Mover targetReference;
     private boolean resetEnabled;

     private BasicPingCell[][] cell;
     private boolean[][] mask;

     private int midMaskRow;
     private int midMaskColumn;

     public PingCellMatrix2(double[] area, double
     detectionRange, double cellSize, double pingInterval,
     Mover searcherReference, Mover targetReference) {

          this.area=area;
          this.detectionRange=detectionRange;
          this.cellSize=cellSize;
          this.pingInterval=pingInterval;
          this.searcherReference=searcherReference;
          this.targetReference=targetReference;
          resetEnabled=true;

          createCellMatrix();
          createMaskingMatrix();
     }

     public double getDetectionRange() {
          return(detectionRange);
     }

     public double getCellSize() {
          return(cellSize);
     }
```

```
public double getPingInterval() {
     return(pingInterval);
}

public double getCurrentPingTime() {
     return(Schedule.simTime());
}

public PingCell[][] getCell() {
     return cell;
}

public void setResetEnabled(boolean enabled) {
     resetEnabled=enabled;
}

public void createCellMatrix() {
     int rows=(int)(Math.rint((2*(area[0]-
     area[1])*2000.0+4*detectionRange)/cellSize)+1);

     int columns=(int)(Math.rint(((area[2]-
     area[3])*2000.0+2*detectionRange)/cellSize)+1);

     cell=new BasicPingCell[rows][columns];

     for(int i=0; i<cell.length; i++) {
          for(int j=0; j<cell[i].length; j++) {
               cell[i][j]=new BasicPingCell(this);
          }
     }
}

public void createMaskingMatrix() {
     int rows=
     (int)(Math.rint(2*detectionRange/cellSize)+1);

     int columns=
     (int)(Math.rint(2*detectionRange/cellSize)+1);

     mask=new boolean[rows][columns];
     midMaskRow=(mask.length-1)/2;
     midMaskColumn=(mask[0].length-1)/2;
     for(int i=0; i<mask.length; i++) {
          for(int j=0; j<mask[i].length; j++) {
               double distance=Math.sqrt(Math
               .pow((i-midMaskRow),2)
```

```
                    +Math.pow((j-midMaskColumn),2));

                    if(distance<=detectionRange/cellSize) {
                         mask[i][j]=true;
                    }
                    else {
                         mask[i][j]=false;
                    }
               }
          }
     }

     public void reset() {
          if(resetEnabled) {
               for(int i=0; i<cell.length; i++) {
                    for(int j=0; j<cell[i].length; j++) {
                         cell[i][j].reset();
                    }
               }
          }
     }

     public void doPing() {
          int midCellRow=(cell.length-1)/2;

          double xSearcher=
          searcherReference.getCurrentLocation()
          .getXCoord();

          double ySearcher=
          searcherReference.getCurrentLocation()
          .getYCoord();

          double yTarget=
          targetReference.getCurrentLocation().getYCoord();

          int yRow=midCellRow+(int)(Math.rint((ySearcher-
          yTarget)*2000.0/cellSize));

          int xColumn=(int)(Math.rint(((xSearcher-
          area[3])*2000.0+detectionRange)/cellSize));

          for(int i=0; i<mask.length; i++) {
               for(int j=0; j<mask[i].length; j++) {
                    if(mask[i][j]) {
```

```
                                cell[yRow-midMaskRow+i][xColumn-
                                midMaskColumn+j]
                                .incrementPings();
                        }
                    }
                }

                this.waitDelay("Ping", pingInterval);
            }

            public void displayCoverage() {
                for(int i=0; i<cell.length; i++) {
                    for(int j=0; j<cell[i].length; j++) {
                        System.out.print
                        (cell[cell.length-1-i][j]
                        .getTimesCovered()+"\t");
                    }
                    System.out.println();
                }
            }
        }
```

```
package swdg;

public class Measures2 {

     public static double[] getMeasures(PingCellMatrix pcm)
{
          double[] measures=new double[1];

          PingCell[][] cell=pcm.getCell();

          int cellsOut=
          (int)Math.rint(pcm.getDetectionRange()/pcm
          .getCellSize());

          int rowInMin=cellsOut;
          int rowInMax=cell.length-1-cellsOut;
          int colInMin=cellsOut;
          int colInMax=cell[0].length-1-cellsOut;

          int midRow=(cell.length-1)/2;
          int columnsCovered=0;
          for(int j=colInMin; j<colInMax+1; j++) {
               if(cell[midRow][j].getTimesCovered()>0) {
                    columnsCovered++;
               }
          }

          measures[0]=(double)columnsCovered/(cell[0]
          .length-2*cellsOut);

          return(measures);
     }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. PSEUDO-CODE FOR A GENERAL BARRIER SEARCH ALGORITHM

```
/*
     LT Wyatt J. Nash
     Operations Research Department
     Naval Postgraduate School
     24 January, 2000

     Pseudo-code for calculating the probability of
     detection for a general barrier search.  A continuous
     stream of targets transit the barrier at a constant
     speed.  The barrier is a fixed area that constrains a
     search asset that is otherwise free to move about in
     the area during the conduct of the search.

     The original implementation of this algorithm was in
     Java 2 so the following pseuo-code closely resembles
     the Java syntax.

     THIS CODE WILL NEITHER COMPILE NOR RUN UNDER JAVA as
     certain functions have been simplified so that they
     may be understood and implemented by anyone with
     knowledge of higher level languages.
*/

//initialize files

     open dataInputFile;
     open dataOutputFile;

//read in search area data

     //initialize the area array
     //area[0]=yMax, area[1]=yMin, area[2]=xMax,
     //area[3]=xMin

     double[] area;
     for (int i=0; i<4; i++) {
          area[i]=dataInputFile.readNextLine();
     }

     //initialize cell size and convert to nautical miles
```

```
        double cellSize=dataInputFile.readNextLine()/2000;

    //read in target data

        //initialize target's speed

        double targetSpeed=dataInputFile.readNextLine();

        //define the time between target exit events

        double targetDeltaTime=cellSize/targetSpeed;
        double targetExitTime=targetDeltaTime;

    //read in searcher data

        //initialize searcher's detection range and convert to
        //nautical miles

        double detectionRange=
        dataInputFile.readNextLine()/2000;

        //initialize the matrix of searcher position data
        //searcherPosition[0]=time of ping
        //searcherPosition[1]=x coordinate
        //searcherPosition[2]=y coordinate

        double[][] searcherPosition;
        while(dataInputFile.hasMoreLines()) {
             for(int i=0; i<3; i++) {
                  searcherPosition[][i]=
                  dataInputFile.readNextLine();
             }
        }

    //create a ping cell matrix of false boolean values

        //calculate number of rows and columns for the ping
        //cell matrix
        //rows and columns are always ODD NUMBERS

        int rows=round((area[0]-area[1]+2*detectionRange)
        /cellSize)+1;

        int columns=round((area[2]-area[3]+2*detectionRange)
        /cellSize)+1;
```

```
     boolean[][] pingCell=new boolean[rows][columns];

//create sensor's masking matrix of false boolean values

     //mask is a square ODD NUMBER BY ODD NUMBER matrix

     int maskSize=round(2*detectionRange/cellSize)+1;

     boolean[][] mask=new boolean[maskSize][maskSize];

     //determine the middle row of the mask

     int mid=(mask.length-1)/2;
     double distance;

     //the outer loop searches the rows of the mask and the
     //inner loop searches the columns of the current row

     for(int i=0; i<mask.length; i++) {
          for(int j=0; j<mask[i].length; j++) {
               distance=sqrt((i-mid)^2+(j-mid)^2);
               if(distance<=detectionRange/cellSize) {
                    mask[i][j]=true;
               }
          }
     }

//the main loop

     //determine how many of the ping cells are outside of
     //the area through which the target may transit

     int cellsOut=round(detectionRange/cellSize);

     //determine the min and max indices of the ping cell
     //matrix that define the area through which the target
     //may tranist

     int colInMin=cellsOut;
     int colInMax=pingCell[0].length-1-cellsOut;

     double x;
     double y;
     int yRow;
     int xColumn;
```

```
        int count;

        //loop through all of the searcher positions

        for(int i=0; i<searcherPosition.length; i++) {

              //allow targets to exit until it is time for
              //another ping

              while(targetExitTime<searcherPosition[i][0]) {

//calculate Pd for exiting target

                    count=0;
                    for(int ii=colInMin; ii<colInMax+1; ii++) {

                          //increment the count if there is a
                          //ping in the cell

                          if(pingCell[0][ii]) {
                               count++;
                          }
                    }

                    //data is the Pd value

                    double data=count/(pingCell[0].length-
                    2*cellsOut);

                    dataOutputFile.write(targetExitTime, data);

//shift the ping cell matrix rows

                    for(int ii=1; ii<pingCell.length; ii++) {
                          for(int jj=0; jj<pingCell[ii].length;
                          jj++) {
                               pingCell[ii-1][jj]=
                               pingCell[ii][jj];
                          }
                    }

//reset the top row of the ping cell matrix

                    for(int ii=0; ii<pingCell[0].length; ii++) {
                          pingCell[pingCell.length-1][ii]=false;
                    }
```

```
                //increment the time at which the next
                //target will exit

                targetExitTime=
                targetExitTime+targetDeltaTime;

            }

//determine the searcher's location and find his
//corresponding row and column in the ping cell matrix

            x=searcherPosition[i][1];
            y=searcherPosition[i][2];

            yRow=round((y-area[1]+detectionRange)/cellSize);
            xColumn=round((x-area[3]+detectionRange)
            /cellSize);

            //update the ping cell matrix for the current
            //ping by scanning the masking matrix for a true
            //value

            for(int ii=0; ii<mask.length; ii++) {
                for(int j=0; j<mask[ii].length; j++) {
                    if(mask[ii][j]) {
                        pingCell[yRow-mid+ii][xColumn-
                        mid+j]=true;
                    }
                }
            }
        }

//close files

close dataInputFile;
close dataOutputFile;
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Koopman, B.G., OEG Report 56, *Search and Screening*, Washington D.C., 1946.

2. Washburn, A. R., *Search and Detection 3rd Edition*, Institute for Operations Research and the Management Sciences, 1996.

3. OPNAVINST 3360.30B, Ship Anti-submarine Warfare Readiness / Effectiveness Measuring (SHAREM) Program, Office of the Chief of Naval Operations (N86), 1994.

4. PEO(USW)INST3360.1A, *USW Measures of Effectiveness*, Program Executive Office for Undersea Warfare, 1995.

5. *Tactical USW Measures of Performance/Measures of Effectiveness (MOE / MOPs)*, Fleet ASW Improvement Program Working Group, 1998.

6. COMURFWARDEVGRU TACMEMO SZ5050-2-93, *Surface Ship Anti-submarine Warfare (ASW) Search Planning*, Commander Surface Warfare Development Group, 1995. (CONFIDENTIAL document)

7. Stork, K.A., *Sensors in Object Oriented Discrete Event Simulation*, Operations Research Dept., Naval Postgraduate School, 1996.

8. McNish, M. J., *Effects of Uniform Target Density on Random Search*, Operations Research Dept., Naval Postgraduate School, 1987.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. John Seeley 2
   Analysis and Technology
   1545 Crossways Blvd., STE A
   Chesapeake, Virginia 23320

4. Commander, Surface Warfare Development Group 1
   2200 Amphibious Drive
   Norfolk, Virginia 23521-2850

5. Commander, Submarine Development Squadron Twelve 1
   Naval Submarine Base New London
   Groton, Connecticut 06349-5200

6. Captain Donald Boland (N-879) 1
   2000 Navy Pentagon, Room 4D542
   Washington, District of Columbia 20350-2000

7. Program Executive Office, Undersea Warfare 1
   2531 Jefferson Davis Hwy.
   Arlington, Virginia 22242

8. Professor James N. Eagle, Code OR/Er 4
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5121

9. Professor Alan R. Washburn, Code OR/Ws 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5121

10. Professor Arnold H. Buss, Code OR/Bu 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93943-5121

11. Wyatt J. Nash 1
1321 W. Lawrence Hwy.
Charlotte, Michigan 48813'